CodeGen Release Notes

V5.3.4 - In Development

CodeGen Release Notes

- We added two new loop constructs called relation loop from key segment loops and relation loop to key segment loops. These loops can be used to iterate through the key segments associated with a relations from and to keys respectively.
- This version of CodeGen was built with Synergy/DE 10.3.3e and requires a minimum of version 10.1.1 to operate.

Symphony Framework Components

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.3.3 - 12th July 2018

- We added the following new structure tokens:
 - <STRUCTURE_NOPLURAL>
 - <STRUCTURE_PLURAL>
- We added the following new field loop tokens:
 - <FIELD_SAMPLE_DATA>
- We added the following new key loop tokens:

- <IF FIRST_UNIQUE_KEY>
- We added the following new relation loop tokens:
 - <RELATION_TOSTRUCTURE_NOPLURAL>
 - <RELATION_TOSTRUCTURE_PLURAL>
 - <IF ONE_WAY_ONE_TO_MANY>
 - <IF ONE_WAY_ONE_TO_ONE>
 - <IF TWO_WAY_ONE_TO_MANY>
 - <IF TWO_WAY_ONE_TO_ONE>
- We added the following new tag loop tokens:
 - <TAGLOOP_FIELD_CSTYPE>
- We added the following new loop utility tokens:
 - <,AND>
 - <,OR>
- We added the following new parameter loop tokens:
 - <PARAMETER_DIRECTION_PAD>
 - <PARAMETER_STRUCTURE_NOPLURAL>
 - <PARAMETER_STRUCTURE_PLURAL>
 - <IF COLLECTION>
 - <IF COLLECTION_ARRAY>

- <IF COLLECTION_HANDLE>
- <IF COLLECTION_ARRAYLIST>
- <IF FIRST_INSTANCE_OF_ENUM>
- <IF FIRST_INSTANCE_OF_STRUCTURE>
- <IF IN_OR_INOUT>
- <IF OUT_OR_INOUT>
- We improved error reporting related to tokens that are found in invalid locations such that the line number and character position of the token are now reported.
- We corrected the following issues that were present in the previous release:
 - An issue that caused the <FIELD_NAME> token to omit the required period between group and group member identifiers in some cases.
 - Issues that caused the <IF COLLECTION> and <IF
 DECIMAL> parameter loop expressions to operate incorrectly.
- This version of CodeGen was built with Synergy/DE 10.3.3e and requires a minimum of version 10.1.1 to operate.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

CodeGen Release Notes

- We added back support for the field and key segment loop expansion token <IF CORECEBOOLEAN> because it broke more templates than expected.
- We added a new structure expansion token <FILE PORTABLE INT SPECS>.
- We added a new structure expression tokens <IF
 FILE_CHANGE_TRACKING>, <IF FILE_COMPRESSION>, <IF
 FILE_PORTABLE_INT_SPECS>, <IF FILE_STATIC_RFA>, <IF
 FILE STORED GRFA> and <FILE TERABYTE>.
- We added a new key segment loop expansion token
 <SEGMENT_ORDER_CODE>.
- We changed the values returned by the <SEGMENT_TYPE> expansion token for auto sequence and auto-timestamp key segments. This change could break your existing template code, but we decided to proceed with the breaking change because in all likelihood the token has not been used yet.
- This version of CodeGen was built with Synergy/DE 10.3.3e and requires a minimum Synergy version of 10.1.1 to operate.

Symphony Framework Components

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.3.1 - 18th May 2018

- We added a new <|> loop utility expansion token.
- We corrected an issue with the <PARAMETER_SIZE> token which will now insert the correct value for the size of structure parameters. The token previously inserted a value of 0 for structure parameters.
- This version of CodeGen was built with Synergy/DE 10.3.3e and requires a minimum Synergy version of 10.1.1 to operate.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.3.0 - 14th May 2018

- We added the ability to generate code based on metadata defined in a Synergy Method Catalog. This includes the introduction of new interface loops, method loops and parameter loops, each with associated expansion and expression tokens. The feature is accessed via new smc and -interface command line options.
- We added several new token case variations to the <KEY_NAME>,
 <SEGMENT_STRUCTURE>, <RELATION_FROMKEY>,
 <RELATION_TOKEY> and <RELATION_TOSTRUCTURE> tokens.
- We significantly enhanced the capabilities of relation loops by adding many new relation loop expansion tokens and relation loop expression tokens.
- We corrected an issue that was causing structure enumeration loops to operate incorrectly.

- We changed the field loop and key segment loop expression <IF
 CORECEBOOLEAN> to <IF_CORECE_BOOLEAN> for improved
 consistency with similar expressions in other loops. This change could
 break your existing template code, but CodeGen will terminate with an
 error message if it encounters the old form of the expression.
- We fixed a crash that would occur if you attempted to declare a userdefined expansion token with the same name as a built-in expansion token.
- We reviewed and re-organized the documentation to make it easier to access information with less clicks.
- This version of CodeGen was built with Synergy/DE 10.3.3e and requires a minimum Synergy version of 10.1.1 to operate.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.2.7 - 6th March 2018

- We fixed an issue that was preventing all loop utility expression
 tokens and also the <REMAINING_EXCLUSIVE_MAX_value> and
 <REMAINING_INCLUSIVE_MAX_value> loop utility expansion
 tokens from operating correctly inside nested loops.
- We made some minor internal changes to prepare for building and deploying CodeGen on non-Windows platforms in the future using .NET Core.
- This version of CodeGen was built with Synergy/DE 10.3.3d and

requires a minimum Synergy version of 10.1.1 to operate.

Symphony Framework Components

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.2.6 - 29th January 2018

CodeGen Release Notes

- We added a new structure expression token <IF
 <p>STRUCTURE_MAPPED that allows you to generate conditional code for mapped structures.
- We corrected an issue that was causing the field loop expression tokens
 IF DATE_YYYYMMDD> and <IF_DATE_YYMMDD> to fail to evaluate as true for nullable date fields.
- This version of CodeGen was built with Synergy/DE 10.3.3d and requires a minimum Synergy version of 10.1.1 to operate.

Symphony Framework Components

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.2.5 - 16th January 2018

- We added four new structure expression tokens <IF
 <p>STRUCTURE_ASCII>, <IF STRUCTURE_ISAM>, <IF
 <p>STRUCTURE_RELATIVE> and <IF
 <p>STRUCTURE_USER_DEFINED>. These expression tokens can make
 it significantly easier to create template files that can create appropriate
 output based on the type of data file that is associated with the structure
 being processed.
- This version of CodeGen was built with Synergy/DE 10.3.3d and requires a minimum Synergy version of 10.1.1 to operate.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.2.4 - 12th January 2018

- We fixed a problem that was occurring when the
 <FIELD_MAXVALUE> replacement token was used in conjunction with decimal or integer fields that use a UI Toolkit Window-script based selection window, and in conjunction with the -ws command line option. The token would previously insert an incorrect value that was equal to the enumerated base value minus the enumerated step value, but without regard to the number of selections present in the window. This behavior has been corrected.
- We Fixed a problem that was preventing UI Toolkit window script files from being parsed correctly.
- This version of CodeGen was built with Synergy/DE 10.3.3d and requires a minimum Synergy version of 10.1.1 to operate.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.2.3 - 1st December 2017

- We added a new experimental utility to the distribution. The Code
 Converter utility can be used to automate bulk searches within and edits
 to an applications code. This utility is in a usable form but is still a work
 in progress and is likely to undergo substantial changes as it evolves.
- We added two new utility routines (IsDate.dbl and IsTime.dbl) that are referenced by some of the supplied sample template files.
- We corrected a regression that was introduced in the previous release which caused the field loop expansion token<FIELD_SQL_ALTNAME> not to default to using the actual field name if no alternate name was present.
- We performed an extensive code review and cleanup, updating the code in several areas to take advantage of new features available in the Synergy compiler, and also improving efficiency.
- We fixed an issue in the CreateFile utility that would result in an unhanded exception in the event that invalid key information was passed to XCALL ISAMC.
- We fixed an issue that was causing the CreateFile utility -r (replace file)
 option to fail, an existing file would not be replaced even if the -r option
 was specified.
- We made some minor code changes to allow CodeGen to be built in a

.NET Core environment and we hope to be able to leverage .NET Core to once again support the use of CodeGen on non-Windows systems (starting with Linux) in the near future.

• This version of CodeGen was built with Synergy/DE 10.3.3d and requires a minimum Synergy version of 10.1.1 to operate.

Symphony Framework Components

- We no longer ship the Symphony Framework sample templates with CodeGen. You can obtain the latest Symphony Framework templates from the Symphony Framework web site.
- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.2.2 - 25th October 2017

- We added a new field loop expansion token
 <FIELD_FORMATSTRING> which can be used to access a fields format string value.
- We added a new command-line option -utpp which instructs CodeGen
 to treat user-defined tokens as preprocessor tokens. This means that
 user-defined tokens are expanded much earlier during the initial
 tokenization phase, which in turn means that other expansion tokens
 may be embedded within the values of user-defined tokens.
- We removed the RpsBrowser utility from the distribution; it was an experimental project that didn't really takeoff.
- This version of CodeGen was built with Synergy/DE 10.3.3d and requires a minimum Synergy runtime version of 10.1.1.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.2.1 - 24th August 2017

- We added several a new generic expression token <IF
 COUNTER_n_op_value> that allow you to write conditional template
 code based on testing the current value of the two internal template
 counters against a simple numeric expression. For example you can test
 whether counter 1 has reached a value of 5 with the expression token
 <IF COUNTER 1 GE 5>.
- We added several new loop utility expression tokens that allow you to write conditional template code based on the total number of items that will be processed by a loop, how many items have already been processed, and how many remain, either including or excluding the current item. The new expressions are <IF

```
PROCESSED_EXCLUSIVE_op_n>, <IF
PROCESSED_INCLUSIVE_op_n>, <IF
REMAINING_EXCLUSIVE_op_n>, <IF
REMAINING_INCLUSIVE_op_n> and <IF TOTAL_ITEMS_op_n>.
An example of using these new tokens is:
```

```
<FIELD_LOOP> <IF TOTAL_ITEMS_LE_100>
   Code for small loops
<ELSE>
   Code for larger loops
</IF TOTAL_ITEMS_LE_100>
```

```
</FIELD LOOP>
```

We added several new loop utility expansion tokens that allow you to
determine the total number of items that will be processed by a loop,
how many items have already been processed, and how many remain,
either including or excluding the current item. The new expressions are

<PROCESSED_EXCLUSIVE>, <PROCESSED_INCLUSIVE>,
<REMAINING_EXCLUSIVE>,
<REMAINING_EXCLUSIVE_MAX_n>,
<REMAINING_INCLUSIVE>,
<REMAINING_INCLUSIVE</pre>
ANAX_n> and <TOTAL_ITEMS>.

- We added a new file header token
 <REQUIRES_CODEGEN_VERSION> that allows you to specify that a minimum version of CodeGen is required in order to successfully process a template.
- We made a minor correction to the documentation of the -f l command line option. The documentation previously stated that this option caused fields marked as "Excluded by Language" to be EXCLUDED from field loop processing, but actually such fields are excluded by default. The -f l command line option actually suppresses this behavior, causing such fields to be INCLUDED in field loop processing.
- This version of CodeGen was built with Synergy/DE 10.3.3c and requires a minimum Synergy runtime version of 10.1.1.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

CodeGen Release Notes

- We added two file loop expansion tokens
 <FLOOP_ODBC_NAME> and <FLOOP_RPS_NAME>.
- This version of CodeGen was built with Synergy/DE 10.3.3c and requires a minimum Synergy runtime version of 10.1.1.

Symphony Framework Components

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.1.9 - 12th May 2017

- We made a slight change to the way that the multiple structures command line option (-ms) is processed, allowing it to be used when only one repository structure is specified. This allows for templates that use the <STRUCTURE_LOOP> construct to be used when only one structure is being processed.
- We fixed an issue that was causing the <FIELD_SPEC> token to
 produce incorrect values for auto-sequence and auto-timestamp fields.
 Previously the value 8 would be inserted, now the correct value i8 is
 inserted.
- We added two new structure expansion tokens
 <FILE_ODBC_NAME> and <FILE_RPS_NAME> that expand to the repository ODBC table name and repository file definition name of the

- first file definition that is assigned to the structure currently being processed.
- This version of CodeGen was built with Synergy/DE 10.3.3c and requires a minimum Synergy runtime version of 10.1.1.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.1.8 - 23rd March 2017

CodeGen Release Notes

- We added a new generic expansion token <<u>DATABASE</u>> that can be used to insert the name of the relational database currently being targeted.
- We added three new generic expression tokens <IF
 <p>DATABASE_MYSQL>, <IF DATABASE_POSTGRESQL> and <IF
 <p>DATABASE_SQLSERVER> that can be useful when developing
 database agnostic template files.
- We enhanced the <REQUIRES_OPTION> file header token by adding a new option TF which allows a template designer to specify that the template-defined folders feature must be enabled.
- This version of CodeGen was built with Synergy/DE 10.3.3b and requires a minimum Synergy runtime version of 10.1.1.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.1.7 - 7th February 2017

- We added experimental support for generating code for MySQL and PostgreSQL databases by adding a new -database command line option. This option allows you to specify a target database type and affects the SQL-compatible data types that are generated by the field loop expansion token <FIELD_SQLTYPE>. If this option is not used then the default database continues to be Microsoft SQL Server, but this default may be changed via the new environment variable CODEGEN_DATABASE_TYPE. Before considering this support final we would appreciate any feedback from developers working with MySQL or PostgreSQL about whether we have chosen appropriate data type mappings. A list of these data type mappings can be found here.
- We changed the SQL Server data type mappings for D6 (YYMMDD) dates from DECIMAL(6) to DATE.
- We changed the SQL Server data type mappings for D6 (YYYYPP) and D4 (YYPP) period numbers from CHAR(6) and CHAR(4) to DECIMAL(6) and DECIMAL(4) respectively.
- We changed the SQL Server data type mappings for nullable time (HHMMSS and HHMM) from DECIMAL(6) and DECIMAL(4) to TIME(0).
- This version of CodeGen was built with Synergy/DE 10.3.3b and requires a minimum Synergy runtime version of 10.1.1.

- Symphony Orchestrator was upgraded to use Symphony Framework V3.2.11.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.1.6 - 7th November 2016

- We modified the way that key loops are processed so that if a repository structure has a mixture of access keys and foreign keys defined, the foreign keys are ignored when processing key loops.
- We added a new key loop expression <IF FIRST_SEG_NOCASE>.
- We added new field loop expressions <IF AUTO_SEQUENCE>, <IF AUTO_TIMESTAMP>, <IF AUTO_TIMESTAMP_CREATED> and <IF AUTO_TIMESTAMP_UPDATED> which can be used to determine if fields are defined as auto sequence or auto time-stamp fields.
- We added new key loop expressions <IF
 AUTO_TIMESTAMP_CREATED> and <IF
 AUTO_TIMESTAMP_UPDATED>.
- We added new key segment loop expressions <IF SEG_AUTO_TIMESTAMP_CREATED> and <IF SEG_AUTO_TIMESTAMP_UPDATED>.
- We changed the behavior of the field loop expansion token
 <FIELD_TYPE_NAME>, which will now report the new values AUTO

SEQUENCE or AUTO TIMESTAMP when auto sequence or auto timestamp fields are encountered. Previously the token would have returned a generic value of INTEGER for both.

• This version of CodeGen was built with Synergy/DE 10.3.3a and requires a minimum Synergy runtime version of 10.1.1.

Symphony Framework Components

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.1.5 - 6th September 2016

CodeGen Release Notes

- We added a new structure expression <IF STRUCTURE_HAS_UNIQUE_PK>.
- We fixed a problem that was preventing loop utility expressions from operating correctly.
- We fixed a problem that was causing the <IF LAST> field loop utility expression to operate incorrectly in primary key blocks.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.

 There were no Symphony Framework CodeGen Extensions changes in this release.

V5.1.4 - 29th July 2016

CodeGen Release Notes

- We changed the SQL data mappings for time (HHMMSS and HHMM)
 fields from DECIMAL(n) to TIME(0). This will result in a change of
 output from the <FIELD_SQLTYPE> token. If this causes you any
 issues you can revert to the previously produced values by using custom
 data type mapping.
- The CodeGen installation was altered so that the changes to PATH
 occur immediately after the installation completes. This means that it is
 no longer necessary to reboot after installing CodeGen on a system for
 the first time.

Symphony Framework Components

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.1.3 - 30th June 2016

CodeGen Release Notes

Added a new loop utility expansion tokens <BSLASH> and <FSLASH>.

- Added a new field loop expression <IF LENGTH_OVER_8>.
- This version of CodeGen is built with Synergy/DE 10.3.3 and targets a minimum Synergy runtime version of 10.1.1.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.1.2 - 28th January 2016

CodeGen Release Notes

- We fixed a bug that was causing the <SELECTION_VALUE> selection loop expansion token to produce incorrect results for enumerated decimal and integer fields. The values being produced were previously off by 1, with values for fields with positive enumerated step values being one too high, and values for fields with negative enumerated step values being one too low.
- We changed the CodeGen installation so that it adds the CodeGen folder to the END of PATH instead of the beginning.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- We fixed a bug that was causing the

<SYMPHONY SELECTION VALUE> selection loop expansion token to produce incorrect results for enumerated decimal and integer fields. The values being produced were previously off by 1, with values for fields with positive enumerated step values being one too high, and values for fields with negative enumerated step values being one too low.

V5.1.1 - 9th December 2015

CodeGen Release Notes

We have moved the CodeGen source code repository from CodePlex to GitHub; the only functional CodeGen changes in this release are related to that move (check for update mechanism, download URL, etc.).

If you don't already have one we encourage you to Create a GitHub Account and to Watch CodeGen. If you wish to receive notifications about new CodeGen releases you can also subscribe tho the CodeGen Releases Atom feed.

We have not changed the licensing terms, CodeGen is still distributed under the terms of the New BSD license (BSD). For the time being we plan to leave the CodePlex environment intact, but no new changes will be checked in there and no new releases will be published there.

Here are a few useful GitHub URLs related to our new home:

Main project https://github.com/Synergex/CodeGen home page

Wiki

(information) https://github.com/Synergex/CodeGen/wiki

Download

https://github.com/Synergex/CodeGen/releases/latest

latest version

Issue https://github.com/Synergex/CodeGen/issues

tracking

Releases https://github.com/Synergex/CodeGen/releases.atom

Atom feed

- We have made an internal change that has consolidated the functionality
 of the CodeGenEngineShared and CodeGenParser assemblies into the
 existing CodeGenEngine assembly and we modified the installation to
 remove the two redundant assemblies.
- We have upgraded the version of the WiX Toolset that we are using to create the CodeGen installation to V3.1.0.2213.

- There were no Symphony Orchestrator changes in this release.
- The Symphony_FileIO template was altered to use
 TAGLOOP_FIELD_SQLNAME> instead of
 TAGLOOP_FIELD_NAME>. This change only affects structures with tags.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.1.0 - 4th December 2015

CodeGen Release Notes

 We added support for the conditional processing of template code based on identifiers defined on the command line.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.0.7 - 20th November 2015

CodeGen Release Notes

- We added support for the conditional processing of template code based on identifiers defined on the command line.
- We added a new structure expression <IF STRUCTURE_HAS_UNIQUE_KEY>.
- We added a new key loop variation called <<u>UNIQUE_KEY</u>>.
- We fixed a bug that was causing a crash during key segment loop processing if a key segment was defined using an overlay field. An appropriate error is now generated.
- We changed the minimum required version of the .NET Framework from 4.5 to 4.5.2.

Symphony Framework Components

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.0.6 - 9th September 2015

CodeGen Release Notes

 We fixed an error that was interfering with the processing of case variations of the field loop expansion token <FIELD_ODBCNAME>.

- We added two new field loop expansion tokens that insert SQL and
 .NET compatible variants of a fields alternate name. The tokens
 are <FIELD_SQL_ALTNAME> and <FIELD_NET_ALTNAME> and
 both are also available for use in key segment loops.
- We added five new field loop expressions <IF ARRAY_FIRST>, <IF
 ARRAY1_FIRST>, <IF ARRAY2_FIRST>, <IF
 ARRAY3_FIRST> and <IF ARRAY4_FIRST>. These expressions can
 be used to determine if a field that was the very first element of an array
 is being processed.

- There were no Symphony Orchestrator changes in this release.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.0.5 - 28th August 2015

- We upgraded the development environment to use Visual Studio 2015.
- We added a new field loop expansion token
 <FIELD_SELECTION_COUNT>.
- We extended the functionality of the recently added checkversion command line option so that you can now configure
 CodeGen to periodically check for updates and inform you when an update is available.
- We have modified CodeGen so that it sets the Windows exit code to 1 if an error occurs during processing. This means that the standard IF

- ERRORLEVEL mechanism can be used to detect code generation problems when operating in batch files.
- We changed the way that field names are generated for fields from multi-dimensioned arrays. In previous versions the name of a field from a multi-dimensioned array might be presented as FIELD1_1_1, but this could result in a problem when using expansion tokens such as <FieldNetName>, or any other pascal-case or camel-case variant of tokens referring to the name of the field. Such tokens suppress the underscore characters and result in names like Field1111. This meant that duplicate names could be generated for fields within the same array. Consider an array field defines as MY_FIELD, [12,12,1]A1; the <FieldNetName> for MY_FIELD[1,11,2] would be MyField1112; exactly the same as for field MY_FIELD[11,1,2]. To address this we have changed the naming convention for elements in multi-dimensioned arrays such that leading zeros are now included, up to the required length based on the highest element number. For example the names produced for the earlier example will now be MyField01112 and MyField11012. If you use expansion tokens that refer to the non-DBL names of fields in multi-dimensioned arrays in such a way that those names become part of an external API then this change will break your existing code. Note that this change only applies to multi-dimensioned arrays; the behavior of fields in single-dimensioned arrays has not changed.
- We fixed a problem with the processing of the tag loop expansion token <TAGLOOP_TAG_VALUE>. Previously for an alpha field with a comparison value the comparison value would have been inserted without trailing space (e.g. "A"), now trailing spaces will be inserted if necessary (e.g. "A"). Previously for an alpha field with an empty comparison value a null string (e.g. "") would have been inserted, now an empty string with the appropriate number of spaces (e.g. "") is inserted. The number of spaces is determined by the length of the alpha field, up to a maximum of 15 characters which is the maximum possible length for a tag comparison value.
- We fixed an error that was preventing the tag loop expression tokens <IF SINGLE_TAG> and <IF MULTIPLE_TAGS> from operating correctly. Previously <IF SINGLE_TAG> would only be true

if no tags were present and <IF MULTIPLE_TAGS> would be true only if a single tag was present.

 We added two new structure expressions <IF STRUCTURE_SINGLE_TAG> and <IF STRUCTURE_MULTIPLE_TAGS>.

Symphony Framework Components

- We corrected the column headings in the Orchestrator commands list.
- There were no Symphony Framework template file changes in this release.
- There were no Symphony Framework CodeGen Extensions changes in this release.

V5.0.4 - 30th June 2015

- **IMPORTANT:** We changed some of the implementation details for period date fields (YYPP and YYYYPP). Previously, because there is no direct equivalent for fields of these types in environments like .NET, Java and SQL, these fields were treated as type **String** by various field loop tokens. We have changed this so that they are now treated as type **int**. This change affects the following field loop tokens:
 - <FIELD_CSTYPE>, <FIELD_CSDEFAULT>, <FIELD_OCTYPE>,
 <FIELD_OCDEFAULT>, <FIELD_SNTYPE>,
 - <FIELD_SNDEFAULT>, <FIELD_VBTYPE> and
 - <FIELD_VBDEFAULT>. This could potentially change the behavior of your templates and the resulting code, but we decided to make the change because fields of this type are fairly rare.
- We added a new -rps command line option which allows the repository main and text files to be used to be specified on the command line, overriding other mechanisms for repository selection such as RPSDAT,

RPSMFIL and RPSTFIL environment variables.

- We added a new -attach command line option which is intended for use by CodeGen developers. When specified this option causes CodeGen to wait for a key press at the very beginning of processing, making it possible to attach an external debugger to the process before processing begins.
- We added a new -time command line option that causes CodeGen to report the total elapsed time taken to process the command.
- We fixed an issue in the repository API which was causing period dates (D4 YYPP and D6 YYYYPP) to be incorrectly reported as type JULIAN instead of DATE. This in turn was causing the field loop expressions <IF DATE_YYPP>, <IF DATE_YYYYPP> and <IF DATE NOT PERIOD> to fail.
- We removed an artificial limitation that was limiting the number of structures that could be processed concurrently (via the -ms command option) to five.
- We corrected a situation where recent versions of CodeGen could potentially exit and report a fail status without having displayed a corresponding error message.
- We fixed issues with the <FIELD_PRECISION> and
 <FIELD_PRECISION2> field loop tokens which were both inserting incorrect values. We also added a new field loop token
 <FIELD_PRECISION0>.
- We fixed an issue that was preventing the field loop expansion token
 <FIELD_SELECTIONS> from operating correctly.
- We fixed an issue that was preventing code generation without a repository structure from working.
- Fixed an issue that was preventing the field loop expressions <IF
 <p>PKSEGMENT> and <IF NOTPKSEGMENT> from operating correctly
 when used with structures that have no keys defined.

We added back several sample subroutines, functions and include files
to the distribution. These files were previously distributed but were
accidentally excluded when we moved to the new WiX based installer.
Some are still referenced by several old sample templates (mainly the
UI Toolkit related templates). The files can be found in the main
templates folder and we also once again set the CODEGEN_INC
environment variable to point to the location of various include files that
are referenced by some of the sample code.

- **IMPORTANT:** When upgrading to this version of CodeGen it it important that you update your Symphony Framework version (via NuGet) to V3.1.10
- In Orchestrator we added the ability to create a new command based on an existing command. The new option is available by right-clicking an existing command in the command grid, and from there you can select the required repository structure and the new commands will be created. The command grid also now allows for the selection of multiple commands.
- Orchestrator now supports being launched via a registered file
 association, and we have updated the installation to create a file
 association for orchestrator project files (.symproj). This means that you
 can now launch orchestrator by double-clicking on a project file, or by
 right-clicking a project file and selecting "Open with Symphony
 Orchestrator".
- Orchestrator also now supports Windows "Jump Lists". This means that
 if you pin the Orchestrator icon to the Task Bar, you can use right-click
 to display and select from a list of recently opened projects.
- There were no changes to the Symphony Framework template files in this release.
- There were no changes to the Symphony Framework CodeGen

V5.0.3 - 16th April 2015

CodeGen Release Notes

- We added a new command line option -checkversion that allows you to check whether a you are running the latest released version of CodeGen. Your current and the latest versions will be displayed, together with a message confirming the status of your installation. This option requires Internet connectivity.
- We updated the installer so that the CodeGen documentation is now an optional component. If the documentation is not installed locally then the codegen -docs command line option will launch the Web-based documentation.
- We changed the location where we install the sample templates, which you will now find in a folder below the public documents folder (C:\Users\Public\Documents\CodeGenTemplates).
- The installation now records the CodeGen version number in the registry. This makes it easier for third-party tools that may depend on CodeGen to determine the installed version.
- Several additional Symphony Framework components are now being distributed as part of the CodeGen installation. See below for more details.

- **IMPORTANT:** If you use CodeGen in conjunction with Symphony Framework then it is important that you should upgrade to Symphony Framework V3.1.9.0 BEFORE installing this version of CodeGen.
- The Symphony Framework CodeGen Extensions are now being distributed as part of the CodeGen installation and have been removed

from V3.1.9.0 of the Symphony Framework installation.

- The Symphony Framework template files are now being distributed with the CodeGen installation and have been removed from V3.1.9.0 of the Symphony Framework installation.
- The folder containing the Symphony template files has changed from C:\Program Files (x86)\Synergex\SymphonyFramework\Template to C:\Users\Public\Documents\SymphonyTemplates, but the environment variable SYMPHONYTPL continues to point to the location of the Symphony templates and should be the primary mechanism used to locate them from the command prompt or batch files.
- No Symphony Framework templates were added, deleted or changed in this release.
- We added a border to the Symphony Orchestrator main window so that the application can be resized for easier use.

V5.0.2 - 16th March 2105

- We corrected several documentation issues where it was stated that several command line options (-s, -t, -a, -ut and others) were limited to a maximum of ten values following the option. This limitation in previous versions of CodeGen was removed in V5.0.0.
- We fixed a bug in the Template Browser utility that was causing it to fail to start if Symphony Framework was not installed.
- We added a new file header token called
 CODEGEN_FOLDER> which allows template files to specify the name of a sub-folder (below the main output folder) where output files should be created. You must use the -tf command line option to enable this feature. By default, if a template file specifies that the output file be

placed in a sub-folder, and a namespace is used within the template, then the folder name is also appended to the namespace being used. The appending of the folder name to the namespace can be overridden by the new -nf command line option.

Symphony Framework Components Release Notes

- The Symphony Framework Orchestrator utility in now being distributed as part of the CodeGen installation has been removed from V3.0.5.0 of the Symphony Framework installation.
- We changed the Orchestrator version number to match the version of CodeGen that it now ships with.
- We mad a change to OrchestratorViewModel to prevent the project folder incorrectly being added to program folders.
- We changed the CommandDetails.xam view to correct the display of the token list cells.

V5.0.1 - 31st January 2015

- Fixed a bug that was causing incorrect data to be output (compared to previous versions) by the <FIELD_NAME> token when the prefix option was used in conjunction with a repository structure that
 contained group fields. The bug was introduced in V5.0.0.
- The current CodeGen version number is now recorded in each assembly via the Assembly File Version attribute.

V5.0.0 - 10th January 2015

Introduction

This release represents a total re-write of the CodeGen code base, embracing

best practice object-oriented development techniques, and utilizing all of the latest and greatest features of Synergy .NET. Starting with this release CodeGen will only be available for use on the Windows platform. This new release has been extensively tested and, despite extensive internal changes, has demonstrated a very high level of compatibility with previous versions. But if you do discover an issue then please provide feedback via the CodePlex Issue Tracking system.

Installing This Version

- If you have an earlier version of CodeGen installed then the installer for this version will perform an upgrade in the usual way.
- If you previously installed a V5 beta release then you must manually remove that version before installing this version.

Symphony Framework Users

In order to use this version of CodeGen you must be using Symphony Framework V3.0.2.0 or higher.

Changes That May Break Your Code

- If you have implemented custom extensions to CodeGen then please be aware that the mechanism for implementing extensions has changed. You will need to implement a new version of your extensions in order to have them work with CodeGen V5. If you need assistance with migrating your extensions to the new environment please contact the CodeGen development team via the General Discussions page on CodePlex.
- When using the in-process CodeGen API and specifying the names of user-defined tokens it was previously necessary to include the < and > characters in the name of the user defined token. V5 does not require the < and > characters to be specified, and in fact will fail to recognize the user defined tokens if you do specify them.
- In earlier versions of CodeGen the sample templates were installed to the folder \ProgramData\Synergex\CodeGEn\Templates. Starting from

this version the sample templates are now installed to the folder \Program Files (x86)\Synergex\CodeGen\Templates.

- We discovered that the <FIELD_ELEMENT> field loop token was not implemented correctly. For non-array fields it was intended to not insert any data into the output stream, but in fact would insert a digit 1. Also, when processing the first element of a single-dimension array field the token would insert 0 instead of 1. Both of these have been corrected, which could change the behavior of your template files.
- We discovered that the <FIELD_ELEMENT0> field loop token was not implemented correctly. For non-array fields it was intended insert a digit 0, but in fact would insert a digit 1. Also, when processing the first element of a single-dimension array field the token would insert 0 instead of 1. Both of these have been corrected, which could change the behavior of your template files.

Other Changes and New Features

This release was primarily about re-developing the internals of CodeGen in order to take advantage of latest technologies, but also to make CodeGen more extensible for future releases. But we also managed to squeeze in a few new features along the way:

- Added new generic expansion tokens <HOST_DNS_NAME>,
 <HOST_IP_ADDRESS>, <RANDOM_10>, <RANDOM_100>,
 <RANDOM_1000> and <RANDOM_INT>.
- Added support for generic expression tokens, and added the first generic expressions <IF DEBUG_LOGGING>, <IF FIELD_PREFIX>,
 <IF FIELD_SUBSET> <IF MULTIPLE_STRUCTURES>, <IF NAMESPACE> and <IF VERBOSE LOGGING>.
- The existing counter tokens and expressions have been re-documented as generic tokens and generic expressions.
- Added support for structure expression tokens, and added the first structure expression tokens <IF STRUCTURE_FILES>, <IF STRUCTURE_KEYS>, <IF STRUCTURE_LDESC>, <IF STRUCTURE_RELATIONS>, <IF STRUCTURE_TAGS> and

<STRUCTURE UTEXT>.

- Added new mechanisms for supporting relation loop expressions, structure loop expressions, and not in loop tokens, although none exist as yet.
- Implemented <IF FIRST>, <IF LAST>, <IF MORE> and <IF
 NOMORE> as loop utility expressions, and removed them from the
 specific individual loops that they were previously supported in.
- The file spec provided within a

 PROVIDE_FILE> file.ext
 PROVIDE_FILE> construct can now include a logical name to specify the location of the file.
- Completed the set of file header tokens that allow you to specify that a
 particular custom expansion token is required by adding
 <REQUIRES_CUSTOM_LOOPUTIL_TOKEN>,
 <REQUIRES_CUSTOM_NOTINLOOP_TOKEN>,
 <REQUIRES_CUSTOM_STRUCTLOOP_TOKEN> and
 <REQUIRES_CUSTOM_TAG_TOKEN>.
- Added a new set of file header tokens that can be used to specify that a
 particular custom expression token is required. For example
 </REQUIRES_CUSTOM_FIELD_EXPRESSION>.
- The <REQUIRES_OPTION>SUBSET</REQUIRES_OPTION> can now be satisfied by either the -subset or -fields command line options.
- When processing multiple structures but NOT using a structure loop, we
 currently support up to five concurrent structures. By default the context
 is set to the first structure specified, and you can switch context to a
 different structure using the <STRUCTURE#1> through
 <STRUCTURE#5> tokens. Currently when using the -ms command
 line option, you must specify between 2 and 5 structures.
- The command line -subset option (process a subset of fields) can now be used in conjunction with the -ms option (multiple structures).
- The command line -fields option was changed to support fields from multiple structures, when processing multiple structures (-ms option). When processing multiple structures you can specify -fields STR1.FIELD1 STR1.FIELD2 STR2.FIELD1 etc. If processing multiple

- structures and you don't specify STR.FIELD in -fields then the first structure is used as a default.
- Added a new -mw command line option. By default, if the same task generates the same output file then it is not written to disk multiple times. This option causes each instance of the file to be written to disk. If you use the in-process interface to CodeGen then the equivalent is to set the new CodeGenTask.MultiWriteFiles property.
- Added a new -dt command line option which creates log files which
 detail the results of the tokenization of template files, and tree output,
 both before and after pre-expansion. The files are created in the output
 folder and are named <template>.tokens.log.
- Added a new -c command line option which causes processing to continue even after errors have been reported. If multiple templates, files or tasks are being processed then this option allows code generation to continue even though one template/structure combination has generated an error. If you use the n-process interface to CodeGen then the equivalent is to set the new CodeGenTaskSet.ContinueAfterError property.
- We have changed the way that CodeGen installations are created. We no longer use InstallShield Limited Edition, but instead are now using WiX (Windows Installer XML), and we're also using WiX Toolset which provides Visual Studio integration.
- The CodeGen installation is now digitally signed by Synergex
 International Corporation. This means that you should no longer see
 Windows SmartScreen warnings when executing an installation that has been downloaded from the Internet.

V4.4.4.1 - 11th September 2014

IMPORTANT: This is likely to be the final CodeGen release that supports building with Traditional Synergy; future versions will only be supported under Synergy .NET. This means that future releases will only be available on Windows. The binary releases and source code for this version will continue to be available from CodePlex so it will be possible to continue to use this version on platforms like Linux and OpenVMS, but future enhancements will

only be available on Windows. Remember that if you are developing on a platform other than Windows, you can generate code on Windows and then move that code to the other platform.

This release contains bug fixes for a small number of issues that were discovered after the V4.4.4 release. Most of these issues only affected the "inprocess" interface to CodeGen that is used by the unreleased "Symphony Orchestrator" utility, but one of the issues also affected the processing of user-defined tokens passed in via the -ut command line option, hence this additional release.

V4.4.4 - 19th August 2014

On OpenVMS we updated the processing of the -i and -o command line options so that a Windows or Unix style period (.) can be used to represent "current directory", in addition to the OpenVMS style [] notation.

We changed the implementation of the -u (user-defined token file) commandline option. Previously all user-defined token files were required to be in the same folder as the template being processed, and were required to have a .tkn file extension. Now user-defined token files can be located anywhere and have any file extension. This change means that when using the -u command line option, you must now specify a full or relative path and file name to the token file to use. Synergy logical names can be used to specify the path.

We fixed a bug that was preventing the various loop counter expressions from working correctly. The bug was introduced in the 4.4.3 release.

We added <IF FIRST> as a new expression in key loops, enum loops, enum member loops, file loops and button loops.

We renamed two tag loop expressions for consistency with other loop structures. <IF FIRST_TAG> becomes <IF FIRST> and <IF LAST_TAG> becomes <IF LAST>.

We added the ability to define a subset of fields to be processed by specifying a list of field names on the command line using the new -fields option, or via the SubsetFields property of CodeGenTask. This enables field subset

processing without having to modify the underlying repository. As with the subset option, this feature can only be used when processing a single repository structure, and can't be used at the same time as the -subset option.

We fixed a problem where when using developer debug mode (-debug) CodeGen would fail to create the developer debug mode log log file if the -r (replace output files) option was not being used.

V4.4.3 - 18th August 2014

Significantly enhanced the CreateFile utility by adding support for the following Synergy V10 (ISAM rev 6) features:

- Auto-sequence keys.
- Auto-time-stamp keys.
- Portable integer specifications.
- Stored GRFA option.
- Track changes option.

CreateFile now ignores any foreign key definitions, and will also now attempt to validate that if key definitions include optional "explicit key of reference" values, those values match the order in which the keys are defined (CreateFile always creates keys based on the order of key definitions in the repository structure, because repository does not have any other reliable mechanism for determining the order of keys in a file).

Significantly enhanced the MapPrep utility. Specifically:

- The default name of new mapped structures was changed from <structure>_NEW to <structure>_MAPPED.
- The name of a new mapped structure can now be specified, overriding the default name. This can be done either via a command-line option -n, or by responding to prompts by using one of the command line options ps or -p. Previously you would have to edit the generated schema file in order to rename the structure from its default new name.

- You can now tell MapPrep whether you want to be prompted for new structure names (-ps), field names (-pf) or both (-p).
- By default MapPrep no longer propagates key definitions into the schema for the new mapped structure because they are generally not required. However there is a new command-line option -k which causes key information to be retained as previously.
- If retaining key information while at the same time renaming fields,
 MapPrep now renames any key segments that relate to fields that have been renamed.
- When renaming structures and fields MapPrep now validates that the names that you provide are valid identifiers, and when renaming fields MapPrep now tracks the names that have been used so far and prevents duplicate field names from being entered later.
- MapPrep now defaults to adding structure and field mappings in the structure and fields long descriptions instead of in the user defined text field, and we have added a new command-line option -u which causes the user defined text field to be used as previously.
- Added the ability to load new mapped structures directly back into the repository; this is achieved by using the new -l command-line option.

Altered the <IF DATE> key segment loop expression to evaluate to true for both nullable and non-nullable dates.

Added two new sample templates that demonstrate how to use SQL Server BULK INSERT to load data into tables very quickly. These new templates are:

- DatabaseRoutinesBulkLoad.tpl
- DatabaseRoutinesBulkLoadMapped.tpl.

Added Repository API support for the ISAM Rev 6 (Synergy 10.1) stored GRFA and change tracking file options.

Added support for ISAM file page sizes of 16384 and 32768 in the following tokens:

- <FILE_PAGESIZE>
- <FLOOP_PAGESIZE>

Added the following new structure tokens:

- <FILE_CHANGE_TRACKING>
- <FILE_STORED_GRFA>

Added the following new file loop tokens:

- <FLOOP_CHANGE_TRACKING>
- <FLOOP_STORED_GRFA>

Added new file loop expressions

- <IF CHANGE_TRACKING>
- <IF NOCHANGE_TRACKING>
- <IF PAGESIZE16384>
- <IF PAGESIZE32768>
- <IF STORED_GRFA>
- <IF NOSTORED_GRFA>

Added the following new key segment loop expressions to bring key segment expressions back into line with field loop expressions.

- <IF DATE_NULLABLE>
- <IF DATE_NOT_NULLABLE>
- <IF SEG_ALPHA>
- <IF SEG_ASCENDING>
- <IF SEG_AUTO_SEQUENCE>
- <IF SEG_AUTO_TIMESTAMP>
- <IF SEG_DECIMAL>

- <IF SEG DESCENDING>
- <IF SEG_NOCASE>
- <IF SEG_SIGNED>
- <IF SEG_TYPE_EXTERNAL>
- <IF SEG_TYPE_FIELD>
- <IF SEG_TYPE_LITERAL>
- <IF SEG TYPE RECNUM>
- <IF SEG_UNSIGNED>
- <IF TIMEOUT>

Changed the <FIELD_VBTYPE> data type mappings for nullable time fields from DateTime, to Nullable(Of DateTime).

Added a new mechanism to allow developers to customize the default data type mappings used with various field loop tokens. This feature is enabled via a new command line option -cdm.

Added a new Template Browser utility that allows you to browse CodeGen template files. The utility displays the sample templates that are shipped with CodeGen, and if Symphony Framework is installed then it will display the Symphony templates also. You can also use the Tools > Options dialog to define the folder where your own user-defined templates are stored and if you do so then the utility will display your own templates too.

V4.4.2 - 27th May 2014

We changed the code that generates SQL compatible names for fields to replace any dollar sign (\$) characters with an underscore character (_).

We fixed a data mappings bug which was causing un-coerced TM4 (HHMM) time fields to be incorrectly be typed as nullable.

V4.4.1 - 29th April 2014

IMPORTANT NOTE: We no longer provide binaries for Synergy/DE V9. For the time being we will continue to make best efforts to ensure that the code base remains compatible with Synergy/DE V9 systems so that you can build your own CodeGen distributions, but at some point that will become more challenging because we want to be able to take advantage of the significant new features available in Synergy/DE V10. If you have not already done so we strongly encourage you to update your development environments to use Synergy/DE V10. Of course of that it not possible right now you will be able to continue to use CodeGen 4.3.5.

Added a new API called MethodCatalogAPI which exposes information about the interfaces, methods, parameters and return types defined in a Synergy Method Catalog. In the future this API will be used to allow code to be generated based on the data from a method catalog, in conjunction with repository based information.

Updated the OpenVMS linker options file (CODEGEN.OPT) and removed the SHS\$SHARE logical name prefix from the references to DBLTLIB and SYNRTL. This change will ensure that CodeGen is linked against the correct Synergy runtime libraries on OpenVMS systems that have primary and alternate installs of Synergy/DE. If you do have primary and alternate installs then you must define system wide logical names for DBLTLIB (translating to SYS\$SHARE:DBLTLIB.OLB) and SYNRTL (translating to SYS\$SHARE:SYNRTL.EXE). These logical names should be defined in your system startup command procedure (SYS\$MANAGER:SYSTARTUP_VMS.COM) immediately after you call the

Synergy/DE startup command procedure (SYS\$MANAGER:SYNERGY STARTUP.COM).

Made some improvements to the build and clean scripts for CodeGen under Traditional Synergy on Windows.

Made some improvements to the BUILD.COM and CREATE_INSTALL.COM command procedures for CodeGen on OpenVMS. Also added a new command procedure named CLEAN.COM to clean files left behind by BUILD.COM and CREATE INSTALL.COM.

Added new field loop expressions <IF DATE_NULLABLE> and <IF DATE_NOT_NULLABLE> which allows you to detect date fields that are coerced to nullable dates.

Changed the <IF DATE> and <IF DATEORTIME> field loop expressions to evaluate to true for dates and nullable dates.

Changed the <IF DECIMAL> field loop expression to evaluate to false for nullable date fields.

Changed the <IF NOTDATE>, <IF NOTDATEORTIME> and <IF NOTDECIMAL> field loop expressions to evaluate to true for nullable date fields.

Updated field loop tokens <FIELD_CSTYPE>, <FIELD_VBTYPE> and <FIELD_SNTYPE> to return appropriate values for nullable date and time fields.

Added a new key segment loop token <<u>SEGMENT_SNTYPE</u>> to insert the Synergy .NET data type for a key segment.

Fixed a bug in CodeGenLauncher where any command line parameter value over 80 characters was causing an invalid subscript exception.

Updated the and improved the CodeGenLauncher mechanism which can be used to initiate code in-process generation, rather than code generation initiated by command-line instructions.

We have started to create some short training videos for CodeGen, and will be creating more as time permits. Videos are currently available for the following subjects:

- Introduction to CodeGen
- Using AUTHOR and COMPANY Tokens
- Introduction to Field Loops
- Using Field Loop Subsets
- Using Template Include Files

Configuring Template Dependencies

You can find the CodeGen Training Videos on YouYube.

V4.3.5 - 2nd February 2014

Improved error processing in CodeGenEngine.CheckStructure(). Also in the same routine we worked around a String.Contains() bug in Traditional Synergy 10.1.1b that was preventing CodeGen from working correctly on non .NET platforms.

General improvements the DatabaseTable sample templates, mainly focused around removing the use of optional parameters and using overload methods instead. These changes should allow relational database code generated from these templates to work in Synergy .NET in addition to traditional Synergy. Also made some minor changes to the same templates that result in code that builds cleaner on OpenVMS systems.

Added some support for auto sequence and auto time stamp key segments (ISAM Rev 6). Updated the possible values returned by the <SEGMENT_TYPE> segment loop token, and added two new key loop expressions <IF AUTO_SEQUENCE> and <IF AUTO_TIMESTAMP>.

Added two new field loop expressions <IF GROUP_EXPAND> and <IF GROUP_NO_EXPAND> which allows you to determine whether or not implicit groups are being expanded to individual fields (i.e. whether or not the -g i command line option is being used).

Added two new field loop tokens that expose the Synergy .NET field data type and default value. The new tokens are <FIELD_SNTYPE> and <FIELD_SNDEFAULT>.

Added a new field loop token <FIELD_NETNAME>. This token behaves identically to the <FIELD_SQLNAME> token, it has been included simply to allow template developers to produce more understandable templates, not having to refer to "SQL" when the template doesn't produce anything that involves the use of SQL.

Resolved an issue with user defined token processing whereby if a user

defined token was used multiple times in the same template file source line, only the first instance would be replaced. Now all instances will be replaced.

When using the -e command line option to echo the command line being used (useful when CodeGen is used from a Windows batch file with echo turned off) we will no longer display a blank line before the command line.

V4.3.4 - 19th December 2013

Corrected a problem where overlay field exclusions and certain other command-line driven field filtering mechanisms were not being honored for group fields.

Fixed a bug in the processing of the field tokens <FIELD_MINVALUE> and <FIELD_MAXVALUE> for implied decimal fields with no digits to the left of the decimal place (e.g. D2.2).

V4.3.3 - 13th November 2013 (Synergy .NET only)

Added three new generic tokens that allow you to generate GUIDs and use them in your generated code. The new tokens are <GUID1>, <GUID2> and <GUID3>. Each time you run CodeGen it will generate new values for the three GUID values exposed by these tokens. These GUID values will only be consistent during a single invocation of CodeGen. The next time you invoke CodeGen new values will be generated. If you need to use the same GUID value in multiple templates then you must process those templates all at the same time during a single invocation of CodeGen. These new tokens are only available when using the Synergy .NET version of CodeGen.

V4.3.2 - 30th October 2013

Removed old tag tokens from several example templates.

Fixed a bug which was causing the default author and company names not to be picked up from the registry under .NET.

Added several additional tag loop expressions: <IF FIRST_TAG>, <IF LAST_TAG>, <IF MULTIPLE_TAGS> and <IF SINGLE_TAG>.

Upgraded to **Synergy/DE 10.1.1b**, Visual Studio 2013 and InstallShield Limited Edition 2013.

V4.3.1 - 7th August 2013

Added support for tag loops. Additional information is available in Tag Loop Tokens and Tag Loop Expressions. As part of this enhancement several structure tokens that previously provided information relating to tags have been removed. The structure tokens that were removed are <TAG_EXPRESSION>, <TAG_EXPRESSION_PATH>, <TAG_FIELD_NAME>, <TAG_FIELD_PATH>, <TAG_FIELD_DEFINE>, <TAG_VALUE>, <TAG_VALUE_DEFINE>, <TAG_END_DEFINE> and <MAPPED_TAG_FIELD>. These tokens were all limited in capabilities because they could only be used with structures that has a single tag expression. You should be able to achieve the same results of all of the removed tokens using the new tag loops feature.

Added a new -lf command line option that causes CodeGen to list the names of the files that were generated. This option is only effective when not using verbose or debug logging (-v or -d) because file names are already listed when using those modes.

Fixed a problem which prevented include file processing <FILE:name.exe> from working in some cases.

V4.2.11 - 18th July 2013

Added several new alternate forms of the <FIELD_SELWND> token to provide template developers better control over the case of field selection window names. Also added a new token <FIELD_SELWND_ORIGINAL> to preserve the case of selection window names in the same way that <FIELD_SELWND> used to.

Enhanced UI Toolkit window script selection window processing (-ws) so that selection window names are no longer case sensitive (they aren't in UI Toolkit). Also the -ws option will now work with selection windows that only have a name but no other attributes specified on the .select line.

Added a new feature which enables some degree of preprocessing of field prompts when a structure is processed by CodeGen. This feature is enabled by setting the CODEGEN_STRIP_FROM_PROMPT environment variable prior to code generation.

Updated the InstallShield LE project to not scan shipped assemblies for dependencies at build time. There appears to be a problem with InstallShield 2012 Spring LE which causes it to incorrectly detect a dependency on Visual C++ V11.0 for ARM processors. This could cause installations for previous versions of CodeGen built for Synergy/DE V10 to incorrectly install the ARM version of the C++ V11 runtime if C++ V11 was not already installed on the system. This could cause a problem if CodeGen for Synergy V10 was incorrectly installed on a Synergy/DE V9 system and then Synergy/DE was upgraded to V10. The Synergy/DE installation would see that a version of C++ V11 was already present (but it was the ARM version), but most Synergy/DE components would fail to execute after installation because of the incorrect C++ runtime. If you encounter this problem then the correct solution is to first delete MSVCP110.DLL and MSVCR110.DLL from your Windows\System32 folder (or your Windows\SysWOW64 folder if you are on a 64-bit system) and then use the Programs and Features utility in Windows Control Panel to perform a "Repair" operation on the 32-bit Synergy/DE installation.

V4.2.10 - 4th June 2013

Added support for user token expressions which allow you to test the presence or absence of a user-defined token. User token expressions are supported in all loop structures.

Added a debug mode logging message to confirm when a different key is being used as a result of using the -opk command line option.

Fixed an issue in <FIELD_PROMPT> which would cause field prompts

longer than 30 characters to be truncated to 30 characters.

Updated several sample templates.

V4.2.9 - 7th February 2013

Altered the behavior of the <FILE:name.ext> and

<FILEIFEXIST:file.ext> template file include tokens. Previously to include an external file into a template file during code generation the file to be included was required to be in the same directory as the template file being processed. Now the default behavior is that the file must be in the current directory, but the location of the file can be overridden by using a logical name within the token. For example if you use the token <FILE:header.def> then CodeGen will expect to find the file in the current directory, but if you use the token <FILE:INC:header.def> then CodeGen will open the file from whatever directory is specified by the environment variable INC. This change was made as a result of a special request from our friends over in the Symphony Framework development team (symphonyframework.codeplex.com).

V4.2.8 - 6th February 2013

Added support for a "second segment restriction" in key loops which can be used to indicate that only the second segment of a key should be processed within a key loop. This can be particularly useful when processing structures that have a structure tag value as the first key segment and the main useful segment of the key is often then the second segment.

Added a new structure token <TAG_VALUE> which inserts the value associated with a structures tag expression. If the structure does not have a tag expression then the token inserts nothing.

Added a new structure token <MAPPED_TAG_FIELD> which allows you to access the name of the first tag field in a mapped structure.

Added a new mechanism to allow developers to override the key that is used within a primary key block. By default the primary key is used within a

primary key block, but the new -opk <keyNum> option allows you to chose to use a different key. This can be useful in rare situations where a master /detail relationship between two structures is based on a relationship from the primary key in the master file to an alternate key in the detail file.

V4.2.7 - 17th December 2012

Added new field loop tokens <FIELD_DIMENSION1_INDEX>, <FIELD_DIMENSION2_INDEX>, <FIELD_DIMENSION3_INDEX> and <FIELD_DIMENSION4_INDEX>. Also added new field loop expressions <IF ARRAY1>, <IF ARRAY2>, <IF ARRAY3> and <IF ARRAY4>. These tokens and expressions are useful when working with multi-dimensioned arrays.

One of the most significant changes in this release is the introduction of automated unit testing within the CodeGen development environment. We are now taking advantage of Visual Studio's built in unit testing framework and and are gradually writing tests for many different areas of CodeGen. At the time of writing we have implemented almost 650 individual tests, and in the process of doing so we have already identified (and fixed) several bugs. Having this automated testing capability in place will help us to perform much more extensive regression testing on a regular basis. And for future new features, developing tests for new features before those new features are developed will help to ensure that we get those new features right first time. Embracing unit testing will help to ensure a consistently high level of quality and reliability for future releases.

Enhanced the CodeGen.RepositoryAPI.Repository class by adding a new property named Enumerations which exposes the enumerations that are declared within a repository database.

Addressed an issue which was causing array and group fields, when expanded by CodeGen, to be listed in the wrong sequence in some cases. Also addressed similar issues with various field loop expressions.

Corrected the output of the <FIELD_MINVALUE> and <FIELD_MAXVALUE> tokens so that the values inserted for integer fields are not preceded with a bunch of leading spaces.

Fixed a problems with the <FIELD_ELEMENT> and <FIELD_ELEMENTO> field loop tokens, which were both inserting incorrect values for non-array fields.

Removed three redundant field loop tokens <FIELD_ARRAYDIMP>, <FIELD_ARRAYDIMPO> and <FIELD_ARRAYDIMR>. Following changes in the way that arrays are processed by CodeGen earlier this year these tokens were no longer useful.

Fixed a bug with the replacement of the generic token <DAY>.

Fixed a bug with the replacement of the field loop token <FIELD_ARRIVEM>.

Fixed bugs with the processing of the <REQUIRES_CUSTOM_RELATION_TOKEN> and <REQUIRES_CUSTOM_SELECTION_TOKEN> token.

Fixed a bug with the <ENUM_NUMBER> token which is documented to return a one-based number but was previously returning a zero-based number.

Fixed a bug in the processing of the <<u>ENUM_MEMBER_COUNT</u>> token.

V4.2.6 - 21st November 2012

IMPORTANT: If you are using CodeGen in conjunction with Symphony Framework then it is important that you do not upgrade to this version of CodeGen until you also upgrade to Symphony Framework V2.1.0.0.

The CodeGen installation on Windows now supports upgrading from a previously installed version. We use Windows Installers "major upgrade" mechanism, which essentially performs an automatic uninstall of a previous version before installing the new version. The earliest version that can be upgraded in this way is V4.1.0 which was released on 6th June 2012. If you are running a version older than 4.1.0 then you will need to manually remove that version one final time before installing this version. You can determine the version of CodeGen that you have installed via the command codegen - version.

Added support for custom expressions in all loop structures where expressions can be used.

Added two sets of counters that developers can use in template files.

Added several counter expressions. Currently, despite the fact that counters can be used anywhere in a template, counter expressions can only be used within one of the existing loop structures. We are investigating the possibility of supporting the use of counter expressions, as well as some other expressions, outside of a loop structure.

Changed the API for adding custom tokens for consistency with the new custom expressions support.

Fixed a problem where unnecessary blank lines were being written to output files. Now if a template line contains a template comment (;//) which does not start in column 1 and is the only thing in the line, or if a template line contains a token but after token replacement the line is blank, blank lines will no longer be output.

Fixed a problem in Repository API which was preventing structures containing nested explicit groups from loading. Also added more developer debug mode logging code in the CodeGenLauncher and CheckStructure routines.

Fixed a problem which occurred if the entire content of a template file was delimited by a level 1 loop (for example a field loop) then the final line of output code could be duplicated in some rare cases. Also fixed an issue where in some cases an output file could include an incorrect additional blank line at the end of the file.

Changed the location where the sample templates are installed on Windows systems. On Windows systems sample templates are now installed in the folder <system_drive>:\ProgramData\Synergex\CodeGen\Templates.

Improved the code in several of the example templates. Various internal enhancements and code cleanup also took place for this release.

V4.2.5 - 10th November 2012

Added the ability to use an alternate form of closing tags for all expressions. For example, if using an <IF ALPHA> expression you can now optionally use the </IF ALPHA> closing tag instead of the generic </IF> closing tag. The behavior is the same in either case, but using the more explicit form can improve readability in complex template files.

Fixed a problem with expressions in nested loops (e.g. key segment loops) where in some cases an expression token from a nested loop would be written to the output file.

Added a custom mechanism to allow BusinessCraft to extend CodeGen by processing their "ProForma" files.

V4.2.4 - 8th November 2012

Added new <FIELD_ROW>, <FIELD_COL>, <PROMPT_ROW> and <PROMPT_COL> field loop tokens.

Added case variations for the <FIELD_ARRIVEM>, <FIELD_CHANGEM>, <FIELD_DRILLM>, <FIELD_HYPERM> and <FIELD_LEAVEM> tokens.

V4.2.3 - 5th November 2012

Added a new generic token <OPTIONAL_USERTOKEN> which can be used to indicate that a template requires a specific user defined token, but which also provides a default value for that user token if a value has not been provided by any of the other mechanisms for providing values for user defined tokens.

V4.2.2 - 3rd November 2012

Added support for optional else clauses in all multi-line expressions. We do not currently support else clauses in in-line expressions, but will investigate

the possibility of adding that feature in the future.

Added a new mechanism to support custom field loop expressions based on testing the presence or absence of text in an input fields user text or long description. For example, if you place the text IGNORE_THIS_FIELD in a fields user text or long description, you can test the presence of that string using the field loop tokens <IF CUSTOM_IGNORE_THIS_FIELD> and <IF CUSTOM_NOT_IGNORE_THIS_FIELD>. Custom field loop expressions should always be typed in upper case in template files. The custom expression text placed in a fields user text or long description is not case sensitive.

Added a large number of new key segment loop expressions to make the expressions available in key segment loops consistent with the expressions available in field loops.

Added a new internal mechanism for initiating code generation via the CodeGenerator, CodeGenTask and CodeGenTaskSet classes. In the future these changes will be used to launch code generation from new desktop applications, including code generation wizards within Visual Studio projects.

Added a new desktop application named CodeGen Task Manager. This application allows you to define task sets which contain one or more code generation tasks, and then replay those task sets to regenerate the resulting code. This application is currently experimental, not all planned functionality has been implemented, and it is not yet fully functional.

Enhanced the error reporting to include the exception details for when CodeGen fails to load a custom tokens assembly. Also various improvements to error logging in other areas.

Changed the way that the setup program for Windows binary releases are created. We used to use InstallShield Express 2012, which is a purchased product. We have now switched to using InstallShield Limited Edition For Visual Studio, which is a free option for Visual Studio 2010 and 2012 developers and still provides the basic functionality that we need.

Tested CodeGen with Synergy/DE 10. For the time being we will be distributing two versions of the CodeGen Windows installation, one built with Synergy/DE 9.5.3b and one built with the latest Synergy/DE 10 beta version. We will stop distributing the 9.5.3b installation when Synergy/DE 10 is

released, but previous builds will continue to be available for download from the CodePlex site.

V4.1.10 - 20th September 2012

Added a new field loop token <FIELD_SELWND> to insert the name of the selection window that is associated with a field, and two new field loop expressions <IF SELWND> and <IF NOSELWND> to allow you to test whether the field has an associated selection window name.

Added a new field loop token <FIELD_ENUMWIDTH> which defines the pixel width associated with an enumerated fields longest enumerated value.

Added two missing imports that were preventing CodeGenEngine from building on OpenVMS.

Improved the SQL SELECT logic in a couple of the sample templates.

V4.1.9 - 7th September 2012

Added a new field loop token <FIELD_NOECHO_CHAR> which is replaced by the character that should be used to represent each character typed into a field, in place of the actual character (think password fields). If the current field being processed is a "no echo" field then the token will be replaced by the no echo character specified in the repository field definition, or by an asterisk (*) if no character is specified. If the field is not a "no echo" field then the token will be replaced by a null string.

Added a new utility program called RpsBrowser. This utility allows you to quickly browse the information available in your Synergy Repository. The utility only has basic functionality at this point and will be enhanced over time.

Added a new feature called key loop first segment restrictions which allow you to process only the first segment of a multi-segment key. For more information refer to the documentation for the <FIRST SEGMENT> token.

Added a new Field Loop Expression <IF NOTDATETODAY> which can be used to detect date fields which do not default to the current date.

Improved the output of <FIELD_MINVALUE> and <FIELD_MAXVALUE> so that a decimal point and trailing digits are not output for non implied decimal fields.

V4.1.8 - 14th August 2012

Added a new field loop token <FIELD_PREFIX> which will be replaced with the field prefix value specified via the -prefix command line option, or with a null string if the -prefix option was not used.

Added a new field loop token <FIELD_BREAK_MODE> and three new field loop expressions <IF BREAK_CHANGE>, <IF BREAK_ALWAYS> and <IF BREAK_RETURN>. Note that the existing <IF BREAK> and <IF NOBREAK> expressions can also be used when processing break fields.

V4.1.7 - 31st July 2012

Added the ability to populate field selection lists based on items defined in UI Toolkit selection windows defined in window script files. For more information refer to Selection Windows.

V4.1.6 - 30th July 2012

Added various new field loop expressions including <IF NOCHECKBOX>, <IF DATE_YMD>, <IF DATE_YMD>, <IF DATE_YYMMDD>, <IF DATE_YYMMDD>, <IF DATE_YYJJJ>, <IF DATE_YYYYJJJ>, <IF DATE_JULIAN>, <IF DATE_NOT_JULIAN>, <IF DATE_YYPP>, <IF DATE_YYYYPP>, <IF DATE_PERIOD>, <IF DATE_NOT_PERIOD>, <IF TIME_HHMM> and <IF TIME_HHMMSS>.

V4.1.5 - 26th July 2012

Added the ability to load custom token extensions which are located in directories other than the main CodeGen directory. This can be done by setting the environment variable (or logical name) CODEGEN_EXTDIR to the directory containing the custom token assembly, ELB or shared image. If CODEGEN_EXTDIR is not found then CodeGen will continue to search for custom token extensions in the main CodeGen directory.

Added a new generic token <REQUIRES_USERTOKEN> which can be used in the header portion of a template file to indicate to CodeGen that the template relies on a particular user-defined token. If the user-defined token has not been specified when attempting to generate code from the template then an error message will be displayed and the template will not be processed.

```
Added new tokens <REQUIRES_CUSTOM_TOKEN>,

<REQUIRES_CUSTOM_FIELD_TOKEN>,

<REQUIRES_CUSTOM_SELECTION_TOKEN>,

<REQUIRES_CUSTOM_KEY_TOKEN>,

<REQUIRES_CUSTOM_SEGMENT_TOKEN>, <REQUIRES_CUSTOM_RE

<REQUIRES_CUSTOM_ENUM_TOKEN>,

<REQUIRES_CUSTOM_ENUM_MEMBER_TOKEN>,

<REQUIRES_CUSTOM_FILE_TOKEN> and

<REQUIRES_CUSTOM_BUTTON_TOKEN>. These tokens allow you to indicate in the header section of a template file that specific custom tokens are required in order to successfully generate code from the template. If CodeGen determines that a required custom token has not been provided then an error message will be displayed and the template will not be processed.
```

Added two new options to the existing <REQUIRES_OPTION> token to allow template developers to specify in a template file that the -prefix or -subset option is required in order to process the template. Also enhanced error logging to report the name of the template being processed when a missing required command line option is encountered.

V4.1.4 - 20th July 2012

Fixed a bug in the <SELECTION_VALUE> token that was causing it to

produce values one lower than required for numeric selection lists.

Fixed a couple of documentation typos.

V4.1.3 - 18th July 2012

Modified the -w command line option to add the ability, when processing based on UI Toolkit input window definitions in a script file, to select the input window to be processed by name as well as by index number.

Added a new -wn command line option which prevents script file input window field attributes from being used to override repository field attributes. This allows you to generate code based on only the fields listed in an input window, but using only the repository field definitions.

V4.1.2 - 17th July 2012

Fixed a bug in the IsNumeric sample functions where they would previously report valid negative decimal and implied decimal numbers as non-numeric.

Fixed a problem where command parameters were not being correctly passed to shell scripts on Unix.

Fixed a bug in %WinDir and improved the build script on UNIX.

Improved %WinDir on UNIX so that it doesn't display "File not found" messages.

Fixed a UI Toolkit Window Script file parsing bug.

Worked around a Synergy .NET bug with alpha -= "VALUE"

V4.1.1 - 19th June 2012

Fixed a bug in the DataMappings.xml file for SQL data mappings of implied

decimal fields. Also renamed the file to DataMappingsExample.xml so that CodeGen doesn't load the file by default but continues to use hard-coded defaults. To customize data type mappings the file should be coped back to DataMappings.xml and then customized.

Enhanced error checking to detect and report missing end of loop tokens for multi-line loops.

Enhanced error checking to better detect if someone mis-types a closing token for an in-line loop.

Added the InstallShield Express 2012 project that is used to build the binary distributions for CodeGen. This content is intended for use by the project coordinator only, but could also serve as a reference for anyone else wishing to build binary distributions of CodeGen using InstallShield Express 2012.

Microsoft has announced that they are removing support for the WIX-based "Installation and Deployment" projects in Visual Studio 2012, so in the future developers will need to select and use a third-party tool (e.g. InstallShield) to build product installations. I don't want CodeGen users to get used to using something that I know is going to be removed in the near future, so I have decided to stop distributing the CodeGenSetup and InstallerCustomActions projects. I will be reverting to using InstallSheild Express 2012 to build the CodeGen binary installations. If developers wish to build binary installations of their own they will need to select a suitable third-party installation product and configure their own installations.

Added a Synergy/DE Workbench workspace and projects to make building for Traditional Synergy on Windows easier for developers who have Workbench installed.

Added a new example template named syn_multi_data_class.tpl which is an example of using structure loops, and also added an example to the documentation.

V4.1.0 - 6th June 2012

We added the ability to process multiple repository structures at the same time

in a single template file. This feature is implemented via a new command line option -ms which instructs CodeGen to make all of the structures specified after the -s command line option available to the template(s) being processed. The default context when starting to process each template is the first structure, but context can be switched to a different structure by using a <STRUCTURE#n> token, where n is the 1-based index of the structure to switch to.

```
We fixed a bug in the processing of the structure token <TAG_EXPRESSION> and added three new tokens named <TAG_EXPRESSION_PATH>, <TAG_FIELD_NAME> and <TAG_FIELD_PATH>.
```

We added some brief documentation for the CreateFile, MapPrep and RpsInfo utilities.

We added a new -file command line option which allows code generation to be initiated based on a repository file definition as a starting point. When using -file all of the structures that are assigned to that file are available to be used in the template, in the same way that you would when using the new multiple structure (-ms) support.

We added support for a new <STRUCTURE_LOOP> capability which allows iteration through any number of structures without prior knowledge of the number of structures to be processed. Structure loops can be used with both multi-structure (-ms option) and file-based (-file option) operations.

V4.0.9 - 4th June 2012

Added the CreateFile utility which allows you to create an ISAM file based on Repository structure, key and file definition information.

Added a new mechanism which allows developers to extend CodeGen by providing custom token processing without the need to edit the core CodeGen source files. For more information refer to Adding Custom Tokens.

Added a new <FILE_LOOP> construct which allows you to generate code for multiple files that are assigned to a single structure. There are several new file

loop tokens and file loop expression tokens, and also support for creating custom file loop tokens.

Added the ability to customize the data type mappings for other languages that are produced by the <FIELD_SQLTYPE>, <FIELD_CSTYPE>, <FIELD_VBTYPE> and <FIELD_OCTYPE> tokens. You can now define custom mappings in an XML file named DataMappings.xml which must be placed in the main CodeGen folder (alongside the executables and libraries). If the XML file is not found then CodeGen will continue to use its default hard-coded data type mappings, but if the file is found then CodeGen will use the mappings defined in the file. For more information refer to Customizing Data Type Mappings.

V4.0.8 - 24th May 2012

Fixed an error with the doc comments for the constructor in the DatabaseTable template.

Added the ability to specify the special @MAP, @MAPF, @UNMAPF, @AUTOINCREMENT, @NOCODEGEN, @CODEGEN_DISPLAY_FIELD, @NOAUDIT and @SUB codes in a fields long description in addition to the fields user text string.

V4.0.7 - 12th May 2012

CodeGen is now published as an open source project on CodePlex. You can download a pre-compiled Synergy .NET binary release of CodeGen, or you can download the source code and build it yourself. The source code download includes scripts that can build CodeGen using Traditional Synergy on Windows, Unix, Linux and OpenVMS systems.

This release includes changes to the SQL Connection API example templates. The old SQL_* templates have been retired, and replaced with two sets of new templates. The templates named DatabaseRoutines*.tpl can be used to generate SQL Connection API code based on subroutines and functions (similar to the old SQL_* templates), and the new DatabaseTable* templates

generate database interaction classes, which are easier to use in "consuming" code.

V4.0.6 - 4th May 2012

IMPORTANT: The default version of CodeGen that is executed by the "codegen" command is now built with Synergy .NET. For the time being the DBR version of CodeGen is still being provided, and can be accessed via the command "codegendbr". Please note that if you have batch files which use "call codegen" commands, you will need to either remove the "call" command, or change your commands to continue to use the DBR version for the time being by changing your batch files to "call codegendbr".

Other changes in this version include:

Added a new key segment loop token <<u>SEGMENT_DESC</u>>.

The MapPrep utility has been enhanced so that it now adjusts the names of key segment fields in line with changes to the underlying field names.

This version of CodeGen is now built with Synergy/DE 9.5.3b

V4.0.5 - 12th April 2012

The CodeGen installation now includes a version of CodeGen built with Synergy .NET. In the near future only the Synergy .NET version will be distributed, but for now it is being distributed for testing purposes so that others can help to validate that it is operating correctly. The Synergy .NET version is currently named CodeGenNet.exe and should operate identically to the Traditional Synergy version of CodeGen. To test with the Synergy .NET version simply use the CodeGenNet command instead of the CodeGen command. Command line options are identical, but please not that if you currently use CodeGen from batch files with a "call codegen" syntax, the call statement is not required when using CodeGenNet.

Added support for processing structures which contain multi-dimensional array fields, including multi-dimensional group field arrays.

Added a new field loop token <FIELD_REGEX> which will produce a regular expression that can be used to constrain and validate user input to a field based on the repository definition of the field.

Added support for processing a named subset of fields in a structure via the subset command line option. Field subsets are defined in the repository by adding syntax like @SUB=name; to one or more fields user text string.

Added a new field loop token <FIELD_INPUT_LENGTH> which can be used to set the maximum number of characters that can be entered into a UI control associated with the field.

Added <IF AUDIT> and <IF NOAUDIT> field loop expressions. [Note the these tokens have since been removed because the same functionality can now be achieved via custom field loop expressions.]

V4.0.4 - 28th February 2012

Added a large number of new field loop expressions.

Added a new field loop tokens <FIELD_ARRAYDIMP0> and <FIELD_ELEMENT0>.

Improved error messages when there is a problem creating an output file.

CodeGen now supports nested groups within a structure, and also single-dimension group arrays.

Added a new -g i command line option which causes implicit groups to be processed as a single field, instead of being expanded out to multiple individual fields. When using this option the field type of the group field becomes a class, and you must independently generate that class.

It is now possible to generate code based on a "pseudo structure" which is made up from the fields in an explicit group (i.e. a group with fields locally defined in a structure). This is only supported for explicit groups which are direct members of the structure, not groups nested within other groups. To generate code based on an explicit group, specify the structure name with -s as

<structure>.<groupField>.

V4.0.3 - 19th February 2012

Added a new -cw command line option to override the default pixel width of a character.

Added a new -ch command line option to override the default pixel height of a row.

Updated the help file to reflect these changes, and also added this release notes page.

V4.0.2 - 14th February 2012

Re-worked group field support such that overlays within groups are processed the same way that non-group overlay fields are processed. Note that nested groups are still unsupported.

Copyright © 2018 Synergex International, Inc.

Introduction to CodeGen

CodeGen is a tool that software developers working in a Synergy Development Environment can use to generate source code. That code might be Synergy DBL code, or might be source code for some other language. CodeGen is not restricted to producing code for any particular development environment or programming language.

Of course you can't just use CodeGen to generate any piece of source code that you can imagine. Before code can be generated a developer must declare the rules that define how a given piece of source code should be created. These rules are defined in template files. CodeGen interprets the code and instructions that a developer has defined in a template file in order to produce useful output.

Defining rules in a template file is only part of the story though. To be really useful CodeGen also needs to have another source of information that defines a context for what is to be generated.

Most software applications revolve around the collection, presentation, manipulation and storage of data. That data is ultimately stored in some type of persistent storage such as in a collection of data files, or in tables in a relational database. When developers work on a particular piece of source code they combine knowledge that they have about an application's data (metadata) with rules or "business logic", in order to create source code that addresses a particular requirement. CodeGen does the same thing.

When developers work in Synergy/DE they have access to an excellent source of metadata that is called the Synergy/DE Repository. A repository holds extensive information about the data structures used by an application, and the attributes of those data structures. A repository can also contain information about the relationships between the various data structures, and even about underlying data storage mechanisms. A Synergy repository is a very rich source of metadata and is the primary source of metadata used by CodeGen.

For developers who use the Synergy/DE UI Toolkit to present their application's user interface there may also be a second source of metadata called Window Script files. In some situations CodeGen can also make use of the metadata contained within these files also.

It's all about metadata! CodeGen takes information about a data structure and combines it with rules that have been defined in a template file in order to create some useful output text.

Basic Principles

Each time a developer uses CodeGen to generate code they typically provide two primary pieces of information:

- 1. A source of metadata, usually the name of a repository structure.
- 2. The name of a template file that defines the rules for what is to be created.

Each time CodeGen generates an output file it usually does three primary things:

- 1. Read information from a metadata source, usually a repository structure.
- 2. Read rules from a template file.
- 3. Merge the rules with the metadata to create an output file.

Video

Here's the Introduction to CodeGen video. Make sure you select a high-definition version of the video, by default YouTube tends to play the lowest resolution which can look pretty awful!

License Agreement

CodeGen is distributed under the terms of the Simplified BSD License. By downloading CodeGen in any form, or by using CodeGen or any of it's components, you accept the terms of that license, which are as follows:

Copyright (c) 2012, Synergex International, Inc.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Template Files

Template files are text files which define some piece of generic source code. They contain a combination of actual source code and tokens that have special meaning to CodeGen. Template files have a .tpl file extension. When CodeGen processes a template file, any tokens that are encountered in the template cause it to take some action. Some tokens cause some piece of information obtained from the metadata source to be written to the output file, while other tokens cause CodeGen to perform some specific action.

A simple template named HelloWorld.tpl might contain:

This template is so simple that it does not require an external source of metadata, as no tokens relating to any specific data structure are being used. The template contains three tokens, <AUTHOR>, <DATE> and <TIME>. You can probably guess what these tokens do:

- 1. The <<u>AUTHOR</u>> token will be replaced by the name of the person using CodeGen.
- 2. The <DATE> token will be replaced by the current date, in MM/DD/YYYY format.
- 3. The <TIME> token will be replaced by the current time, in HH:MM format.

If this template were processed by CodeGen an output file would be created and would contain containing something like:

```
;;; Description: A Synergy function that returns "Hello World"
;;
;; Author: Jodah Veloper
;;
;; Created: 03/11/2010 at 11:23
;;
function HelloWorld, a
endparams
proc
freturn "Hello World"
endfunction
```

Default Output File Names

The name of the output file created by CodeGen would be helloworld.dbl. This is because CodeGen has default rules for the naming of output files. If no repository structure is being used then the default output file name is the base name of the template being used, with a file extension of .dbl, in lowercase:

```
<template>.dbl
```

If a repository structure is being used then the default output file name is the name of the repository structure, followed by an underscore, followed by the base name of the template, with a file extension of .dbl, all in lowercase:

```
<structure>_<template>.dbl
```

Customizing Output File Names

You can override these default output file naming rules by adding a special <CODEGEN_FILENAME> token to the template file which defines rules for naming the output file to be created. For example the template file could be updated like this:

```
<CODEGEN_FILENAME>HelloWorldFunction.dbl</CODEGEN_FILE
;;
;; Description: A Synergy function that returns "Hello World"
;;
;; Author: <AUTHOR>
```

```
;;; Created: <DATE> at <TIME>
;;
function HelloWorld, a
  endparams

proc

freturn "Hello World"

endfunction
```

If the template was then reprocessed the name of the output file would be changed to HelloWorldFunction.dbl.

The content of the output file would be exactly the same as in the earlier example, because the <CODEGEN_FILENAME> token is a special instruction to CodeGen and does not cause anything to be sent to the output file.

Template File Comments

Template files can contain special comments which can be used to document the template file, but which are also not sent to the output file. A template file comment is any text which follows the special sequence of characters;//. For example, the template file could be updated like this:

```
function HelloWorld, a
endparams

proc
freturn "Hello World"
endfunction
```

Using Repository Metadata

Template files which do not require a source of metadata are rare, because it is difficult to generate anything useful without a source of metadata! Most templates are written to create output based on a source of metadata, which is usually a repository structure.

Most of the useful metadata is found in repository structure and field definitions, but some tokens require you to define keys, tags, relations and files in the repository also. For more information, refer to Repository Setup.

For a simple example of how a repository structure can be used as a source of metadata, consider the following hypothetical structure definition

```
STRUCTURE DEPARTMENT

DEPT_ID ,A10 ;Department ID (primary key)

DEPT_NAME ,A30 ;Department name

DEPT_MANAGER ,D4 ;Managers employee ID

ENDSTRUCTURE
```

If a programmer wanted to create a template for a Synergy external subroutine which would accept the value of the primary key for a record and then read and returned the record, they could write a template like this:

```
<CODEGEN_FILENAME>Get<StructureName>.dbl</CODEGEN_FILE
;//<CATEGORY>Synergy Subroutines</CATEGORY>
;//<DESCRIPTION>Creates a subroutine that returns a record
from a file</DESCRIPTION>
;//-------;// Template author: Jodah Veloper
```

```
;// Revision: 1.0
             03/11/2010
;// Date:
;; Description: Returns a <STRUCTURE_NAME> record
;; Author: <AUTHOR>
;;
;; Created: <DATE> at <TIME>
.include "<STRUCTURE_NAME>" repository,
structure="str<StructureName>", end
subroutine Get<StructureName>
  <PRIMARY_KEY>
  <SEGMENT_LOOP>
  required in a<SegmentName>, <segment_spec>
  </SEGMENT_LOOP>
  </PRIMARY_KEY>
  required out a < StructureName >, str < StructureName >
  endparams
  stack record
    ch<StructureName>
                         ,int
  endrecord
proc
  <PRIMARY KEY>
  <SEGMENT LOOP>
  a<StructureName>.<segment_name> = a<SegmentName>
  </SEGMENT LOOP>
  </PRIMARY KEY>
  try
  begin
```

```
open(ch<StructureName>=syn_freechn(),i:i,"<FILE_NAME>")
read(ch<StructureName>,a<StructureName>,keyval(ch<StructureName>,
    end
    catch (ex)
    begin
    clear a<StructureName>
    end
    finally
    begin
     if (chopen(ch<StructureName>))
        close ch<StructureName>
    end
    end
    endtry
    xreturn
```

Having created the template, the developer could then process the template in conjunction with the repository structure DEPARTMENT, and CodeGen would create an output file like the one below. Notice that the template file includes a <CODEGEN_FILENAME> rule, so in this case the name of the output file would be GetDepartment.dbl. The file would contain:

```
;;; Description: Returns a DEPARTMENT record
;;
;; Author: Jodah Veloper
;;
;; Created: 11/28/2014 at 12:00
;;
.include "DEPARTMENT" repository, structure="strDepartment", end
subroutine GetDepartment
  required in aDeptId, a10
  required out aDepartment, strDepartment
```

```
endparams
  stack record
    chDepartment
                        int,
  endrecord
proc
  aDepartment.dept_id = aDeptId
  try
  begin
    open(chDepartment=syn_freechn(),i:i,"DAT:department.ism")
read(chDepartment,aDepartment,keyval(chDepartment,aDepartment,0))
  end
  catch (ex)
  begin
    clear aDepartment
  end
  finally
  begin
    if (chopen(chDepartment))
       close chDepartment
  end
  endtry
  xreturn
endsubroutine
```

Notice how all of the tokens that were present in the original template have been replaced by meaningful pieces of information from the metadata source, resulting in a useful piece of source code. We could just as easily use the same template to create a routine to read an employee record, or a location record, or a customer record. The idea is to define the rules in a template once, then use that template to generate many routines.

Sample Templates

CodeGen ships with a collection of sample template files. that are intended to provide you with examples of some of the types of output files that you can create with CodeGen, but you may be able to use some of the templates unaltered.

Having installed CodeGen you will find the sample templates in the Public Documents folder, which us usually in this location:

C:\Users\Public\Documents\CodeGenTemplates

You can also use the Template Browser utility to view the sample templates.

The sample templates may also be viewed directly on the GitHub web site at https://github.com/SteveIves/CodeGen/tree/master/SampleTemplates.



Expansion Tokens

Expansion tokens are special symbols that can be used within a template file, either to cause CodeGen to take some special action, or to refer to some information from a metadata source. Most expansion tokens cause CodeGen to output some piece of information to the output stream for the current template. In some cases the information that is output is fairly generic, for example the current date or time, but in many cases the information is derived from an external source such as a repository structure. CodeGen recognizes and processes several different categories of expansion tokens. Some of these tokens can be used at any time, whilst others can only be used within a certain context in a template file.

Expansion tokens consist of a single tag (a special keyword delimited by lessthan (<) and greater-than (>) characters) and most cause some value to be written to the output stream for the current template. An example of an expansion token of this type is the <<u>DATE</u>> token that was discussed earlier.

When CodeGen processes a template file in order to create an output file it is constantly searching for known expansion tokens, and when a token is encountered some action is taken based on the name and nature of the token.

Generic **Tokens**

Generic tokens are tokens which can be used in any template file.

Structure **Tokens**

Structure tokens are tokens which are replaced with some piece of information obtained from, or some piece of code derived from a Repository structure. Structure tokens can only be used when processing a template in conjunction with a Repository structure. Processing a template file which uses one or more structure tokens without referencing a Repository structure will cause an error during code generation.

Tokens

Field loop tokens are tokens which are replaced with some piece of information obtained from, or some piece of code derived from a Repository field definition. Field loop tokens are only valid Field Loop inside a field loop. This again implies that a Repository structure is required in order to process a template containing field loop tokens. Processing a template file which uses one or more field

loop tokens without referencing a Repository structure will cause an error during code generation.

Field Loop Expression Tokens

Field loop expression tokens are tokens which allow you to control whether a certain piece of code is emitted to the output file or not, based on the evaluation of some expression based on the attributes of field. For example, you could use a conditional expression to only emit a piece of code if a field is marked as a required field, or if the data type of a field is decimal.

Key Loop Tokens

Key loop tokens are tokens which are replaced with some piece of information obtained from, or some piece of code derived from a Repository key definition. Key loop tokens are only valid inside a key loop.

Key Segment Loop Tokens Key segment loop tokens are tokens which are replaced with some piece of information obtained from, or some piece of code derived from a Repository key segment definition. Key segment loop tokens are only valid inside a key segment loop.

Relation Loop Tokens Relation loop tokens are tokens which are replaced with some piece of information obtained from, or some piece of code derived from a Repository relation definition. Relation loop tokens are only valid inside a relation loop.

Window Tokens Window tokens are tokens which are replaced with some piece of information about a window definition.

Button Loop Tokens Button loop tokens are tokens which are replaced with some piece of information relating to a button definition. Button loop tokens are only valid within a button loop.

User-Defined Tokens User defined tokens are non-standard tokens that can be defined by you in order to have CodeGen generate code other than what it natively supports. There are several ways of defining user defined tokens. User defined tokens do not inherently require the use of a Repository structure during code generation, unless the code generated by the user defined tokens includes other structure or field loop tokens.

Loop	Loop utility tokens insert information based on the processing
Utility	status of the current template file loop construct, and can be used
Tokens	in any of the loop constructs.

Expansion Token Variations

All expansion tokens are valid in upper case (as shown in this documentation) and many are valid only in upper case. Generally, when the name of an expansion token appears in uppercase, then the associated replacement text that is written to the output stream will also be in upper case. But there are some exceptions to this rule, typically when the token refers to a piece of metadata which resolves to a piece of "free text" such as descriptions that were entered as free text.

Some tokens are documented to have "alternate forms". For example, the <<u>STRUCTURE_NAME</u>> token, which normally outputs the name of the repository structure in all uppercase characters, has five alternate forms, as follows:

Token Type	Description	Example
Upper- <structure_name case<="" td=""><td>This token causes the name of the current structure to be sent to the outpufile, in all uppercase, and with underscores preserved.</td><th>t CUSTOMER_CONTACT</th></structure_name>	This token causes the name of the current structure to be sent to the outpufile, in all uppercase, and with underscores preserved.	t CUSTOMER_CONTACT
_	This token causes the name of the current structure to be	
Lower- <ase <a="" href="mailto:structure_name"><ase< a=""></ase<></ase>	sent to the outpu file, in all	t customer_contact

lowercase, and with underscores preserved.

This token causes the name of the current structure to be sent to the output file, in

lowercase, but

Customer_Contact

with the first letter of each word uppercased, and with underscores preserved.

This token causes the name of the current structure to be sent to the output file, in lowercase, but

with the first letter of the first word uppercased, and with underscores

preserved. This

compatible with the way in which **GENCS** and **GENJAVA**

expose Synergy

format is

data to xfNetLink Customer_contact

"xf"-

<Structure_name> case

Mixed- <Structure_Name>

case

clients.

This token causes the name of the current structure to be sent to the output file in "pascalcase". The string is lower-cased and then any character following CustomerContact anything except 0-9, A-Z and a-z is upper-cased. Then anything other than 0-9, A-Z and a-z is removed, and finally the first character is upper-cased.

Camel- <structureName> case

Pascal- <StructureName>

case

This token causes the name of the current structure to be sent to the output file in "camelcase". The string is lower-cased and then any customerContact character following anything except 0-9, A-Z and a-z is upper-cased. Finally anything

other than 0-9, A-Z and a-z is removed.

If an expansion token is not documented to have alternate forms then that token is ONLY valid when typed in upper case. CodeGen will not act upon tokens that it does not recognize. Any token that CodeGen does not explicitly recognize will simply be ignored, and written to the output stream. If you see tokens in your output file, check your spelling and case sensitivity. The only exception to this rule is if you are using "user defined tokens" which are documented later.

Copyright © 2018 Synergex International, Inc.

Expression Tokens

Expressions are special tokens that can be used within template files in order to include or exclude a piece of code based on some condition. There are several different types of expressions that can be used in different places within a template file. For example, there are expressions that can be used in a field loop which allow you to include or exclude sections of code based on the attributes of the field that is currently being processed.

There are two types of expressions:

Multi-line expressions cause one or more lines of output code to be included or excluded, based on the evaluation of the expression, and occur when the opening and closing conditional tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
<IF expression>code </IF[ expression]>
```

In-line expressions cause part of an output line to be included or excluded based on the evaluation of the expression, and occur when the opening and closing conditional tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
[code] <IF expression> code </IF[ expression]> [code]
```

Alternate Expression Closing Tags

In complex template files you may prefer to use an alternate format of the closing </IF> tag which includes the name of the condition. For example, if you are using an <IF INTEGER> expression you can chose to specify the closing tag as </IF INTEGER>. Doing so can help to make template files more readable.

Expression Else Clauses

Else clauses are supported in multi-line expressions, like this:

```
<IF expression> code
```

```
<ELSE>
code
</IF[ expression]>
```

And in in-line expressions, like this:

```
[code] <IF expression> code <ELSE> code </IF[ expression]> [code]
```

Nesting Expressions

Multi-line expressions may be nested within one another. For example:

```
<IF expression1>
[code]
<IF expression2>
code
</IF[ expression2]>
[code]
</IF[ expression1]>
```

In-line expressions may be nested within multi-line expressions.

```
<IF expression1>
[code]
[code] <IF expression2>code</IF[ expression2]> [code]
[code]
</IF[ expression1]>
```

Multiple in-line expressions may be used within the same line:

```
[code] <IF expression1>code</IF[ expression1]> [code] <IF
expression2>code</IF[ expression2]> [code]
```

In-line expressions may be embedded within one another:

```
[code] <IF expression1>[code] <IF expression2>code</IF[
expression2] > </IF[ expression1] > [code]
```

Copyright © 2018 Synergex International, Inc.

Conditional Processing

Conditional processing blocks provide the ability to include or exclude parts of a template file based on the presence or absence of identifiers that can be defined using the -define command line option. Unlike expression tokens the evaluation of conditional processing expressions a does not change based on the current context within a template file, rather it remains consistent throughout the entire code generation job.

Conditional processing is based on identifiers that can be defined on the command line. For example:

```
codegen -s CUSTOMER -t FILEIO_CLASS -define ATTACH IO HOOKS -r
```

And is used to include or exclude parts of the code in a template file by evaluating the presence or absence of the identifier via special <IF DEFINED_identifier> and <IF NOT_DEFINED_identifier> tokens. For example:

```
(IO_OPEN_UPD),
begin
open(a_channel=0,u:i,"<FILE_NAME>") [ERR=openError]
<IF DEFINED_ATTACH_IO_HOOKS>
new <StructureName>Hooks(a_channel)
</IF>
end
```

There are two types of conditional processing block:

Multi-line conditional processing blocks cause one or more lines of code to be included or excluded based on the presence or absence of the defined identifier and occur when the opening and closing conditional tags appear on separate lines in a template file. They delimit one or more entire lines of template code, like this:

```
<IF [NOT_]DEFINED_identifier>code </IF>
```

In-line conditional processing blocks cause part of a line to be included or excluded and occur when the opening and closing conditional tags appear on the same line in a template file. They delimit part of a line of template code, like this:

```
[code] < IF [NOT_]DEFINED_identifier > code </IF > [code]
```

Conditional Processing Block Else Clauses

Else clauses are supported in multi-line conditional processing blocks, like this:

```
<IF [NOT_]DEFINED_identifier>
code
<ELSE>
code
</IF>
```

And in in-line conditional processing blocks, like this:

```
[code] <IF [NOT_]DEFINED_identifier> code <ELSE> code
</IF> [code]
```

Nesting Conditional Processing Blocks

Multi-line conditional processing blocks may be nested within one another. For example:

```
<IF DEFINED_identifier1>
[code]
<IF DEFINED_identifier2>
code
</IF>
[code]
</IF>
```

In-line conditional processing blocks may be nested within multi-line conditional processing blocks.

```
<IF DEFINED_identifier1>
```

```
[code]
[code] <IF DEFINED_identifier2> code </IF> [code]
[code]
</IF>
```

Multiple in-line conditional processing blocks may be used within the same line:

```
[code] <IF DEFINED_identifier1> code </IF> [code] <IF DEFINED_identifier2> code </IF> [code]
```

In-line conditional processing blocks may be nested within one another:

```
[code] <IF DEFINED_identifier1> [code] <IF
DEFINED_identifier2> code </IF></IF> [code]
```

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

Installing & Configuring CodeGen

Requirements

CodeGen is a Synergy .NET application and is always built with the latest version of Synergy that is available at the time that a CodeGen release takes place. To install CodeGen your system must have:

- Microsoft .NET Framework 4.5 (or higher) installed.
- Synergy/DE runtime components (both 32-bit and 64-bit if installing on a 64-bit system).
- Access to a Synergy runtime license (e.g. RUN10).

Downloading CodeGen

CodeGen can be downloaded from CodePlex as either a pre-built Windows Installer package, or as a source code package that you must build yourself. You can access these downloads here:

- Windows Installer Package
- Source code package

Installing CodeGen

If you downloaded the Windows Installer package then simply execute the installer and respond to the various prompts. We recommend accepting the default responses to most questions, which will perform a full product installation. We also recommend entering your actual name and company name when prompted, because the responses to these questions are used to provide values for the AUTHOR and COMPANY tokens, and so could appear in source code that you may generate.

System Environment PATH

For the best experience the main CodeGen installation folder should be included in the system PATH.. By default the CodeGen installer takes care of

this.

Setting Environment Variables

There are several environment variables that you can set in order to configure an appropriate environment for CodeGen. These environment variables are discussed below.

You can set these environment variables in one of the following ways:

- 1) In your actual system environment.
- 2) In your current process environment.
- 3) In the [codegen] section of your synergy.ini or synuser.ini file.

Environment Variable	Comments	
CODEGEN_TPLDIR	Determines the default location of template files. If you don't set this environment variable then CodeGen will default to the "Templates" folder below the main folder that you installed CodeGen into. This folder contains sample templates that ship with CodeGen.	
	You can override CODEGEN_TPLDIR by using -i command line option.	
CODEGEN_OUTDIR	Determines the default location where output files generated by CodeGen who be created. If CODEGEN_OUTDIR is not set then CodeGen will default to creating output files in the current directory.	
	You can override	

CODEGEN_AUTHOR

CODEGEN_COMPANY

CODEGEN EXTDIR

CODEGEN_OUTDIR by using the -command line option.

Determines the value inserted into generated code when the <AUTHOR> token is processed. Who you install CodeGen you are prompte to enter your name and this value is stored in the Windows Registry and used to provide a default value for CODEGEN_AUTHOR. You can override this behavior by explicitly setting CODEGEN_AUTHOR.

If CodeGen is unable to determine the name of an author then the name "CodeGen" is used.

Determines the value inserted into generated code when the <COMPANY> token is processed. When you install CodeGen you are prompted to enter your company nam and this value is stored in the Windov Registry and used to provide a defaul value for CODEGEN_COMPANY. You can override this behavior by explicitly setting CODEGEN_COMPANY.

If CodeGen is unable to determine the name of an author then the name "Synergex" is used.

If you have created custom extension for CodeGen then this environment variable allows you to specify the folder where CodeGen will search for custom extensions assemblies.

SYNDEFNS

The SYNDEFNS environment variab can be used in your Synergy development environment to determin the default namespace that items are added to when performing strong prototyping. CodeGen will also use the SYNDEFNS environment variable to determine the default value to be used when the <NAMESPACE> token is processed.

You can override this default behavio by using the -n command line option.

Developers sometimes encode information into repository field prompts and then pre-process the prompts at runtime before they are displayed within a UI Toolkit application. One common example is when developers want to display field numbers in UI Toolkit input windows In this case, prompts in the repository might be configured as "xx Custome name", and the developer could then execute code to replace the xx characters with field numbers base on the fields position within an input window or set. This can be a problem when using the field definitions in other environments, so CodeGen has CODEGEN_STRIP_FROM_PROMPT the ability to strip a known pattern of characters from any field prompt whe that pattern is found. For example, if some or all field prompts are prefixed with the characters xx<space> then you can tell CodeGen to remove these characters by setting the following environment variable before code

generation:

set
CODEGEN_STRIP_FROM_PROMF
= "xx "

NOTE: you must delimit the text to t stripped with two double quote characters, otherwise CodeGen will n make any changes to field prompts.

Specifies the type of relational database that code is being generated for, by customizing the values inserte by the field loop expansion token <FIELD_SQLTYPE>. Supported values are SQLServer, MySQL and PostgreSQL. Also refer to the command line option -database, whic if used will override any value set by CODEGEN_DATABASE_TYPE. Without either of these options being used the default database type is SQLServer.

CODEGEN_DATABASE_TYPE

Rebooting After Installation

The CodeGen installation modifies the system wide PATH environment variable, and some versions of Windows require a reboot for this new setting to take effect. We recommend that you reboot your PC after installing CodeGen for the first time.

Updating CodeGen

Recent versions of the CodeGen installation support upgrading to a later version, so in most cases an older version of CodeGen will be automatically removed when you attempt to install a newer version. It should not be necessary to reboot your system after an upgrade.

Uninstalling CodeGen

CodeGen can be removed from your system via an Uninstall shortcut which is added to the Windows Start Menu, or via the "Programs and Features" application in the Windows Control Panel.



Command Line Reference

To generate code with CodeGen a developer goes to the operating system command prompt in an appropriately configured environment and issues codegen commands.

As a minimum the developer must use the -t option to specify the name of at least one template file to be used, and in almost all situations the -s option is also used to specify one or more repository structures to be processed.

There are several other command line options documented below. Some of the main ones are the -r option which causes any existing output files to be replaced, and the -v option which produces various information messages to be displayed as CodeGen generates code.

For example, a developer might use a command like this:

C:> codegen -t data_class -s customer -r -v

This command would cause CodeGen to create an output file based on the code defined in the template file named data_class.tpl and using information from the repository structure named CUSTOMER. If the output file already exists then it will be overwritten, and verbose messages will be displayed.

codegen <options>

Commonly Used Command Line Options

These options are used almost every time CodeGen is used:

-s structure[.group] [...] | *

Structure(s) to process. You must specify the name of at least one structure and may optionally specify multiple structure names separated by spaces. Alternatively you can specify the * wild-card to indicate that all structures are to be processed. It is also possible to generate a "pseudo structure" based on the fields within an implicit group within a structure.

In-process API equivalent: CodeGenTask.Structures property

(List<String>).

-t [*]template [...]

Template(s) to process. You must specify at least one template name, and may optionally specify multiple template names separated by spaces. Alternatively you can use the wild-card character * to specify that multiple templates should be processed (e.g. database*) or use the * wild-card alone to indicate that all templates should be processed.

In-process API equivalent: CodeGenTask.Templates property (List<String>).

-r

Replace existing files. By default existing files are not replaced.

In-process API equivalent: CodeGenTask.ReplaceFiles property (Boolean)

Other Command Line Options

These options can be used to modify how CodeGen works:

-a alias [...]]

Structure aliases. A structure alias is an alternate name to be used to represent a structure. If -a is used then you must specify at least one alias name, and may optionally specify multiple alias names separated by spaces, up to but not exceeding the number of structure names that have been specified via the -s option.

In-process API equivalent: CodeGenTask.Aliases property (List<String>).

-ba|d|n

Override default button loop processing rules.

-ba Always use the default buttons defined in

DefaultButtons.xml.

-bd Never use the default buttons defined in DefaultButtons.xml.

-bn Never load any buttons (DefaultButtons.xml or window script).

In-process API equivalents:

- -ba CodeGenTask.AlwaysLoadDefaultButtons property (Boolean)
- -bd CodeGenTask.NeverLoadDefaultButtons property (Boolean)
- -bn CodeGenTask.NeverLoadAnyButtons property (Boolean)

-C

Continue processing after errors. If multiple templates, files or tasks are being processed then this option allows code generation to continue even though one template/structure combination has generated an error.

In-process API equivalent: CodeGenTaskSet.ContinueAfterError property (Boolean)

-cdm <filespec>

Specifies that custom data mappings are loaded from an XML file. Refer to Customizing Data Type Mappings in the docs for more info.

In-process API equivalent: CodeGenTaskSet.DataMappingsFile property (String)

-ch <pixels>

Overrides the default number of pixels per row to be used when calculating pixel coordinates using tokens like <FIELD_PIXEL_ROW>.

In-process API equivalent: CodeGenTask.CharacterHeight

property (int)

-checkversion [interval]

Checks whether there is a later version of CodeGen available for download. This option requires Internet connectivity. If you do not specify an interval value then CodeGen will perform an immediate check for available updates and will report the result. If you specify an interval of 1 or more (days) then CodeGen will automatically check for available updates after that many days has expired since the previous update check; the update check will take place the next time you use CodeGen and you will see messages as the update check takes place. To cancel a previously scheduled automatic update checks specify an interval of 0.

-cw <pixels>

Overrides the default number of pixels per character to be used when calculating pixel coordinates using tokens like <FIELD_PIXEL_WIDTH>.

In-process API equivalent: CodeGenTask.CharacterWidth property (int)

-d

Debug mode. Displays additional information during processing; more than verbose mode.

In-process API equivalent: CodeGenTaskSet.LoggingLevel property (use LoggingLevel.Debug)

-database SQLServer | MySQL | PostgreSQL

Database type. Specifies the relational database that code is being generated for by altering the SQL data types emitted by the <FIELD_SQLTYPE> token. The default database type is SQLServer, unless altered by the CODEGEN DATABASE TYPE environment variable.

In-process API equivalent: CodeGenTaskSet.DatabaseType property

-define <identifier> [<identifier>...]

Defines one or more identifiers, the presence of which can be tested in template files to implement conditional processing.

In-process API equivalent: CodeGenTaskSet.Defines property (List<string>)

-docs

Displays the CodeGen documentation, either the Windows help file if the documentation is installed locally, or the on-line documentation if not. Bear in mind that the on-line documentation will always be for the latest version of CodeGen, which may be later than the version that you are currently running.

In-process API equivalent: None

-dt

Debug tree. Creates log files containing the internal results of the code tokenization and parsing phases.

In-process API equivalent: CodeGenTaskSet.LogTokenizerResults property (Boolean)

-е

Echo the command line used to start CodeGen. This can be useful when starting CodeGen from a batch file in which echo has been disabled but where you want to see a representation of the commands that are being executed.

In-process API equivalent: CodeGenTaskSet.EchoCommands property (Boolean)

-f [o|l|t|r|w]

Override default field loop processing rules.

- -f o Include overlay fields.
- -f l DO NOT exclude fields which are "Excluded by Language".
- -f t Exclude fields which are "Excluded by Toolkit".
- -f r Exclude fields which are "Excluded by ReportWriter".
- -f w Exclude fields which are "Excluded by Web".

In-process API equivalents:

- -f o CodeGenTask.IncludeOverlayFields property (Boolean)
- -f l CodeGenTask.IgnoreExcludeLanguage property (Boolean)
- -f t CodeGenTask.HonorExcludeToolkit property (Boolean)
- -f r CodeGenTask.HonorExcludeReportWriter property (Boolean)
- -f w CodeGenTask.HonorExcludeWeb property (Boolean)

-fields [str.]field [[str.]field] ...

Creates a field subset based on the list of fields listed on the command line. When processing multiple structures concurrently (via the -ms or -file options) field names may be prefixed with the appropriate structure names.

In-process API equivalent: CodeGenTask.SubsetFields property (List<Tuple<String, String>>)

-file filename

Repository file name to process. This option causes all structures that are assigned to the file definition to be available for use when processing templates. The end result is similar to specifying multiple structures with -s and using the -ms option.

In-process API equivalent: CodeGenTask.RepositoryFile property (String).

-fo file [...]

File Overrides. Allows you to specify which repository file definition is used for each structure being processing. This option

is useful if the structure that you are using is assigned to multiple files because by default CodeGen will use the first assigned file. When the -fo option is used you must specify at least one repository file definition name, and have the option of specifying file overrides for multiple structures, up to but not exceeding the number of structures specified via the -s option.

In-process API equivalent: CodeGenTask.FileOverrides property (List<String>).

-g f|r|i

Override default group field name prefix rules

- -g f Don't prefix group fields with group name.
- -g i Don't expand implicit groups to individual fields.
- -g r Don't use repository group field prefix.

In-process API equivalents:

- -g f CodeGenTask.GroupFieldNoGroupPrefix property (Boolean)
- -g i CodeGenTask.GroupNoExpand property (Boolean)
- -g r CodeGenTask.GroupFieldNoRpsPrefix property (Boolean)

-h | -?

Display command-line help information.

In-process API equivalent: None

-i inpdir

Input file directory containing templates. Defaults to the current directory, or the directory specified with the CODEGEN_TPLDIR environment variable. You can specify a full or relative path, or a logical name followed by a colon (e.g. TPL:).

In-process API equivalent: CodeGenTaskSet.TemplateFolder property (String) or CodeGenTask.TemplateFolder property (String)

List Files. Lists the names of any files generated when not running in verbose or debug logging modes.

In-process API equivalent: CodeGenTaskSet.ListGeneratedFiles property (Boolean)

-ms

Multiple Structures. Specifies that the structures that follow the -s command line option should all be available to templates being processed, at the same time.

In-process API equivalent: CodeGenTask.MultipleStructures property (Boolean).

-mw

Multi-write a file if the same file is generated more than once.

In-process API equivalent: CodeGenTask.MultiWriteFiles property (Boolean)

-n namespace

The namespace to use when replacing the <NAMESPACE> token. Codegen will determine the default namespace using the environment variable SYNDEFNS, and this can be overridden with the -n option.

In-process API equivalent: CodeGenTask.Namespace property (String)

-nf

Don't append sub-folder to namespace. If a template file uses the <CODEGEN_FOLDER> token to specify that files created from the template should be created in a sub-folder below the main output folder, then by default, if a namespace is also being used,

the name of the sub-folder is appended to the namespace. This option overrides that behavior.

In-process API equivalent: CodeGenTask.DontAppendFolderToNamespace property (Boolean)

-o outdir

Output directory. Defaults to the current directory, or the directory specified with the CODEGEN_OUTDIR environment variable. You can use a full or relative path, or a logical followed by a colon (e.g. SRC:).

In-process API equivalent: CodeGenTaskSet.OutputFolder property (String) or CodeGenTask.OutputFolder property (String)

-opk <keyNumber>

Overrides the key that is used when a primary key block (<PRIMARY_KEY>) is used. By default the first key defined by the structure is used.

In-process API equivalent: CodeGenTask.PrimaryKeyNumber property (int)

-prefix <name>

When using the <FIELD_NAME> token, prefix field names with <name>. Useful with repository .include with the optional prefix= qualifier.

In-process API equivalent: CodeGenTask.FieldPrefix property (String)

-rps <mainfile> <textfile>

Allows the repository main and text files to be used to be specified on the command line, overriding other mechanisms for repository selection such as RPSDAT, RPSMFIL and RPSTFIL environment variables.

In-process API equivalent: CodeGenTask.RepositoryMainFile and CodeGenTask.RepositoryTextFile properties (String)

-subset name

Subset of fields to use. Specify subsets in field user text strings with @SUB=name; syntax. This option is only valid when processing a single repository structure.

In-process API equivalent: CodeGenTask.Subset property (String).

-tf

Allow template defined folders. From CodeGen V5.0.2 it has been possible for template files to define the name of a sub folder (below the main output folder) where files generated from the template should be created. This is done by using the file header token <CODEGEN_FOLDER>. This option enables this functionality.

In-process API equivalent: CodeGenTask.AllowTemplateFolder property (Boolean)

-u usertokenfile

User defined token file. Specify a full or relative file spec of the file that contains user defined token definitions. The file spec can include Synergy logical names.

In-process API equivalent: CodeGenTask.UserTokenFile property (String)

-ut name=value [...]

User defined tokens specified via the command line. You must specify at least one name=value, and may optionally specify multiple name=value pairs separated by spaces. If spaces exist in the name or value then you should enclose the entire name=value pair in quotation marks.

In-process API equivalent: CodeGenTask.UserTokens property (List<UserToken>)

-utpp

Treat user-defined tokens as preprocessor tokens, expanding them out to their constituent parts during the initial tokenization phase. This enables other tokens to be embedded in the values of user-defined tokens.

-V

Verbose mode. Displays additional information during processing; not as much as debug (-d) mode.

In-process API equivalent: CodeGenTaskSet.LoggingLevel property (use LoggingLevel.Verbose)

-version

Display the version of CodeGen that you are running.

-w <wscfile>[#n]

Process input from a UI Toolkit Window script file. By default the fields and buttons collection will be based on the first input window found in the script file, but this can be overridden using the #n qualifier, which specifies either a 1-based input window number or the name of the input window to process.

In-process API equivalent: CodeGenTask.WindowScript property (String)

-wn

When processing input from a UI Toolkit Window script file this option prevents script file input window field attributes from being used to override repository field attributes. This allows you to generate code based on only the fields listed in an input window, but using only the repository field definitions.

In-process API equivalent:
CodeGenTask.IgnoreScriptFieldAttributes property (Boolean)

-ws <wscfile>

Window script selection list processing. When this option is used CodeGen will examine any fields which have selection windows specified and will attempt to populate the field selections property based on the selection items defined in the named selection window.

In-process API equivalent: CodeGenTask.SelectionWindowScript property (String)

Processing a Synergy Method Catalog

These options are used to perform code generation based on metadata contained in a Synergy Method Catalog XML export file.

-smc <smcXmlFile>

Synergy method catalog XML file to use. When using this option you may not use the -s option to specify specific structures to process. If you also specify the -interface option then only that interface will be processed, and you may not use interface loops in your template files. If you do not specify the -interface option then you MUST use an interface loop in your template file.

In-process API equivalent: None.

-interface <interfaceName>

The name of the interface in the Synergy Method Catalog XML file that you wish to generate code for. This option is only valid in conjunction with the -smc option, refer to the details of that option above for additional information.

In-process API equivalent: None.

Advanced Debugging Options

These options are intended for use by CodeGen developers and are not particularly useful when using CodeGen to generate code.

-attach

Intended for use by CodeGen developers, this option causes CodeGen to wait for a key press at the very beginning of processing. This makes it possible to attach an external debugger to the process before processing begins.

In-process API equivalent: None

-debug

Developer debug mode (displays extensive debugging information).

In-process API equivalent: CodeGenTaskSet.LoggingLevel property (use LoggingLevel.DeveloperDebug)

-pause

Causes CodeGen to wait for a key press at the very end of processing. This makes it possible to perform final tasks before a debugging environment terminates.

In-process API equivalent: None

-time

Displays the elapsed time taken to process the command.

In-process API equivalent: None

Preparing Your Repository

CodeGen is almost always used with a repository structure providing the metadata that is to be used for the generation of the source file, and many of the tokens require the use of a repository structure.

The basic requirement is that you have a structure definition, and that structure definition includes one or more field definitions. Some tokens also require that keys are defined, and some require file definitions with structure assignments.

If you don't already have a repository then you can load most of the information that you will need by reading existing record layout files into repository structures.

Repository Structure Restrictions

In order to be used successfully with CodeGen, Repository structures must meet the following requirements.

- Enumeration, binary and structure fields are not currently supported by CodeGen.
- Overlay fields are allowed, but will generally be ignored by CodeGen.
- All keys must be correctly defined to exactly match the appropriate ISAM file.
- Only a single tag is allowed and that tag must be a FIELD.EQ.VALUE type.
- The use of some tokens requires that the structure being processed is assigned to a file definition, and some require that keys are defined.
- Keys must be defined in the correct sequence. CodeGen determines key numbers by the order in which keys are defined in the repository structure. Define access keys in the correct sequence, and define foreign keys after all access key definitions.
- If you intend to generate source code for languages other than Synergy,
 then you should avoid giving fields names which may be reserved

words in the language concerned.

- If you wish to prevent a structure from being processed by CodeGen then you can add the text @NOCODEGEN to the structures user text field or long description.
- To specify that a field in a structure is the structures "display field" you can add the text @CODEGEN_DISPLAY_FIELD to the fields' user text string or long description. For more information refer to the <DISPLAY_FIELD> token.

Array Fields

CodeGen supports array fields, but these arrays are "flattened" into multiple non-array fields. In other words each array field encountered is replaced by multiple fields with the same data type and size as the array field.

When an array is expanded to multiple individual fields, the name of each field is amended to include the array element definition. For example, the field MYFIELD, [2]A10 will be expanded to two separate fields named MYFIELD[1] and MYFIELD[2].

Group Fields

CodeGen supports groups, including group arrays. When processed by CodeGen, groups are "flattened". In other words the fields in the group become fields in the main structure, and the group field is removed.

By default, group member field names are prefixed with the name of the parent group field followed by a period, but this behavior can be overridden with the –g f command line option. For example, if a group called ADDRESS contains a field called STREET, the name of the field used by CodeGen will be:

ADDRESS.STREET

But if you were to use the –g f command line option then the name of the field would be:

STREET

Obviously, in this scenario you would need to make sure that the name did not clash with any other identifiers in the same structure.

If the repository group definition includes a "member prefix" then this prefix is added to the name of each field in the group, but this behavior can be overridden with the –g r command line option.

For example, if a group called ADDRESS contains a field called STREET, and the group field defines a member prefix of HOME_, the name of the field used by CodeGen will be:

ADDRESS.HOME_STREET

With the -g f command line option the name would be:

HOME_STREET

With the -g r command line option the name would be:

ADDRESS.STREET

And with the -g f r command line options the name would be:

STREET

If a group member field is declared as an array then it will be treated as discussed earlier under Array Fields.

Loading Include Files

Before you load the fields from an include file into a repository structure it is often best to make a copy of the include file, and then perform some simple "cleanup" within the file before loading it into the repository structure:

- The repository can load the fields from a single record (or common) in an include file into a repository structure, so if the include file has multiple records defined then first identify which record you wish to load.
- Repository will use any comment which follows a field definition as the description for the field, so clean up the comments. If your field

comments include things like position information then you may want to remove them. Note that if the positions are delimited with specific identifiable characters (for example if they are within parentheses) then you can specify these characters when you load the fields into repository, so in this case you don't need to manually edit them out.

- Repository will load any overlays that are defined within the record to be loaded, but if you have overlays that you want to preserve that are defined as separate overlay records (using record, X) then you might want to consider moving those overlay fields into the actual record definition.
- Remember that generally CodeGen will ignore overlay fields however, so review any overlays and make sure that the fields that you are most likely to want to use in generated code are the "real" fields, and those less likely to be useful are the overlays. An example would be if you have date fields where the real fields are individual year, month and day, and there is a full date overlay. When generating code it is most likely that you will want to access the full date and not the individual fields, so switching the fields and overlays around would probably be best.
- If you have full overlay records (one record followed by one or more record, X's) then you will probably want to load each of the overlays into a separate repository structure.

Once you have cleaned up the include file you are ready to load the file into a new repository structure. The procedure is as follows:

- Start repository and Select Modify > Structures.
- Click the Add button (or press the Insert key) to add a new structure.
- Provide a name for the structure. It is usually best to use the original record name as the structure name.
- Set the file type. Most structures should be set to "DBL ISAM".
- Provide a meaningful description of the structure, such as "Customer Master Record".
- If the structure is to be associated with a file which has multiple record layouts within the data file, set the tag to tell repository and other tools

how to detect this particular record layout within the file. If this is not the case then select None.

- Click the OK button to create the new structure.
- Click the Attributes button (ore press F3)
- From the menu, select Attributes > Fields.
- From the menu, select Field Functions > Load Fields (or press F9).
- Enter the name of the include file (or use the file browser to locate it).
- If the include file has multiple records, enter the number of the record to be read.
- If you want to strip information (for example position between parentheses) from the field descriptions then enter the characters that delimit the information to be removed.
- Click the OK button to load the fields into the structure.
- Press the F4 key to exit from the fields list.
- Press the F4 key again to exit from the structure, and save the changes.
- Keep pressing F4 to exit from the repository program.

Copyright © 2018 Synergex International, Inc.

Structure Aliasing

Structure aliasing is a mechanism which allows you to apply an alternate name (or alias) to a repository structure. Aliasing provides a mechanism to be able to generate code with more meaningful names, regardless of legacy names which may be present in your repository or data definitions. For example, your customer master file record layout might be defined by a repository structure named CUSMAS, but you may wish to refer to the structure by an alias of CUSTOMER when generating code.

When you specify an alias for a structure that alias will be used whenever the name of the repository structure would normally be used. For example the <STRUCTURE_NAME> token will be replaced by the name of the alias that you have provided instead of the name of the actual structure being used.

Specifying Structure Aliases

Structure aliases are specified via the -a command-line option. You can specify up to ten aliases after the -a option, based on the number of structures that you have specified after the -s command line option.

For example, if you are processing a single repository structure your command line might look something like this:

```
codegen -s CUSMAS -a CUSTOMER -t DataClass -r
```

If you were processing multiple structures, and wanted to apply an alias to each, you might use a command like this:

codegen -s CUSMAS ORDMAS ORDLIN -a CUSTOMER ORDER_MASTER ORDER_LINE -t DataClass -r

Accessing the Underlying Structure

Swapping out the structure name for a specified alias is exactly what you want in most cases, but there is one important exception. If you are trying to access the actual underlying repository structure, then using the alias name isn't going to work. For example if you have aliased your CUSMAS structure as CUSTOMER then the following template code:

```
.include "<STRUCTURE_NAME>" repository,
structure="str<StructureName>"
```

Would produce the following output file code:

.include "CUSTOMER" repository, structure="strCustomer"

Clearly the problem is that there is no CUSTOMER structure in the repository, it's just an alias for CUSMAS, so the resulting code won't compile.

The solution to this problem is provided by the <STRUCTURE_NOALIAS> token which is always replaced by the name of the underlying repository structure, even if a structure alias has been specified. When writing templates you should ALWAYS use this token when you are trying to access an actual repository structure. For example:

```
.include "<STRUCTURE_NOALIAS>" repository, structure="str<StructureName>"
```

Would produce something output file code like this:

.include "CUSMAS" repository, structure="strCustomer"

Copyright © 2018 Synergex International, Inc.

Structure Mapping

There are occasions when it is useful to be able to process more than one repository structure at the same time, or at least to have the ability to make references to a second structure, or fields in the second structure.

For example, consider the situation where you are creating routines which you will use to access data in a relational database table. One of the routines that you create might be responsible for creating a relational database table based on the attributes of a repository structure (fields become columns, etc.). You may find that the repository structure does not quite meet the requirements of what you are trying to achieve. For example, you may want to assign different names to the columns it the database table, omit some fields from the table, or perhaps change some data types. You can't change the current repository structure to meet your new requirements, because that would impact your other applications, and obviously the current structure needs to match the layout of the actual data file.

To assist in addressing issues like these, CodeGen allows you to create a new repository structure and then "map" the new structure to an existing structure. You can also map fields in the new structure to fields in the original structure.

To map a new structure to an existing structure you add a special code into the new structures long description or user text field, like this:

To map fields in your new structure to fields in an existing structure, add @MAP={otherfield}; to the new fields long description or user text string, like this:

Note that in both cases it is important that the word MAP appears in uppercase, and there should be no spaces between the leading @ character and the trailing; (semi-colon) character.

Creating Mapped Structures

To assist you in creating mapped structures in the repository, CodeGen

includes a utility program called MapPrep. This utility can significantly reduce the amount of effort required to create mapped structures, especially when the structure has a large number of fields.

Mapped Field Conversions

When performing field mapping, you may optionally specify functions to be called when data is mapped into the field in the new target structure, or unmapped back into the field in the original structure. This can be useful if you need to perform transformations on the data as mapping occurs. For example, you may wish to convert the data in a description field between uppercase in the original field, and mixed case in the new field, or convert between a decimal field in the original structure and an implied decimal in the new structure.

To specify a mapping conversion function for a mapped field you add another special code in the fields' user text string or long description, like this:

```
@MAPF=mapping_function;
```

To specify a mapping un-conversion function for a mapped field you add another special code in the fields' user text string or long description, like this:

```
@UNMAPF=unmapping_function;
```

Writing Conversion Functions

Mapping and unmapping functions must be declared as external functions, with a return value data type which matches the target field. They must accept a single parameter which is used to pass in the original value to be mapped or unmapped. For example:

```
function DivideBy100, d. required in originalValue, d endparams

proc
freturn (originalValue / 100.0)
endfunction
```

Copyright © 2018 Synergex International, Inc.

Field Subsets

Field subsets is a mechanism which allows you to define one or more subsets of fields within a repository structure, and then generate code based on a subset of the structure which contains only the fields that have been declared as a member of that subset.

To declare that a field is a member of a subset, add text like the example below to the fields user text string:

@SUB=name;

Where "name" is the name of the subset that you wish to define. For example, if you wish to create a subset called "COMBO" in a structure, and have that subset contain two fields, then you would add this text to the user text string for both of those fields:

@SUB=COMBO;

Once you have a subset defined, you can cause CodeGen to include only the fields in that subset by using the -subset command line option. For example:

codegen -t DataClass -s CUSTOMER -subset COMBO

Field Subsets From Multiple Structures

If you want to be able to process a subset of fields from multiple structures then you can combine the -ms command line option with the -subset command line option.

Command Line Subsets

Another way to define a subset of fields to be processed is to use the -fields command-line option. With this option you can name the fields that you want to be part of the subset directly on the command line, without the need to modify the repository. For example:

codegen -t DataClass -s CUSTOMER -fields FIELD1 FIELD2 FIELD3

Command Line Subsets With Multiple Structures

It is also possible to specify a subset of fields when processing multiple structures. For example, you could use a command like this:

codegen -t DataClass -s STR1 STR2 -ms -fields STR1.FIELD1 STR1.FIELD2 STR2.FIELD1 STR2.FIELD2

Video

Here's the Using Field Loop Subsets video. Make sure you select a high-definition version of the video, by default YouTube tends to play the lowest resolution which can look pretty awful!

Copyright © 2018 Synergex International, Inc.

Selection Windows

If a particular field in a repository structure has selection items associated with it, those selection items can be specified in one of two ways.

- 1. The items can be defined in a selection "List" directly within the repository field (or template) definition.
- 2. The items can be defined in a UI Toolkit selection "Window", which is defined externally in a window script file and referenced by name in the repository field (or template) definition.

When using the first of these two mechanisms the selection list data is available to CodeGen and can be used to generate code using a selection loop within a field loop.

However, when a repository field refers to an external selection window, the information about the fields selections is not automatically available to CodeGen. But you can make the information available by naming the window script file containing the referenced selection windows via the -ws command line option.

When using the -ws option bear in mind that it may be the case that several fields in the repository structure may reference different selection windows, and those selection windows may be defined in different window script files. This problem can be compounded if multiple repository structures are being processed at the same time.

The -ws option only allows you to name a single window script file, so if you find that you concurrently need to access selection windows that are defined in multiple window script files then we suggest that you temporarily concatenate all of your window script files into a single file and then name that file with the -ws option.

IMPORTANT: If you do concatenate multiple window scripts into a single file, remember that it may then be possible to have duplicate selection windows present in that file! In this situation CodeGen will use the first selection window that it encounters. It is strongly recommended not to have multiple selection windows with the same name!

Copyright © 2018 Synergex International, Inc.

File Header Tokens

File header tokens are a special class of token that provides high level processing information to the code generator. None of the file header tokens cause any output to be written directly to the output stream of the current template. All file header tokens consist of a pair of matching opening and closing tags that surround some significant value. Any file header tokens that you use in a template file must appear together at the very top of the template file.

The file header tokens are:

File Header Token	Description
<codegen_filename></codegen_filename>	Allows you to define the name of the output file within a template file.
<codegen_folder></codegen_folder>	Allows you to specify a subfolder (below the main output folder) in which the output file should be
	created. Allows you to specify that a template
	requires a specific user defined token, but allows
<optional_usertoken></optional_usertoken>	you to provides a

<PROCESS_TEMPLATE>

<PROVIDE_FILE>

<REQUIRES_CODEGEN_VERSION>

default value for that user token if a value has not been provided by any other mechanism.

Allows you to specify that an additional template is to be processed immediately after the current template.

Allows you to specify that an additional file is also provided to the output directory as part of the processing of the current template.

Allows you to specify that a minimum version of CodeGen is required in order to successfully process the

<REQUIRES_CUSTOM_BUTTON_EXPRESSION>

template.

Allows you to specify that a particular custom button loop expression is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_BUTTON_TOKEN>

Allows you to specify that a particular custom button loop token is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom enumeration expression is required in order to be able to successfully generate code

<REQUIRES_CUSTOM_ENUM_EXPRESSION>

from the template.

Allows you to specify that a particular custom enumeration loop token is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom enumeration member loop expression is required in

required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom enumeration member loop

<REQUIRES_CUSTOM_ENUM_TOKEN>

<REQUIRES_CUSTOM_ENUM_MEMBER_EXPRESSION> required in

<REQUIRES_CUSTOM_ENUM_MEMBER_TOKEN>

token is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_FIELD_EXPRESSION>

Allows you to specify that a particular custom field loop expression is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_FIELD_TOKEN>

Allows you to specify that a particular custom field loop token is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular

<REQUIRES_CUSTOM_FILE_EXPRESSION>

custom file loop expression is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom file loop token is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom key loop expression is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_FILE_TOKEN>

<REQUIRES_CUSTOM_KEY_EXPRESSION>

<REQUIRES_CUSTOM_KEY_TOKEN>

Allows you to specify that a particular custom key loop token is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_LOOPUTIL_EXPRESSION>

Allows you to specify that a particular custom loop utility expression is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_LOOPUTIL_TOKEN>

Allows you to specify that a particular custom loop utility token is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_NOTINLOOP_EXPRESSION>

Allows you to specify that a particular custom "not in loop" expression is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_NOTINLOOP_TOKEN>

Allows you to specify that a particular custom "not in loop" token is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_RELATION_EXPRESSION>

Allows you to specify that a particular custom relation loop expression is required in order to be able to successfully generate code

from the template.

Allows you to specify that a particular custom relation loop token is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom key segment loop expression is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom key segment loop token is required in

<REQUIRES_CUSTOM_RELATION_TOKEN>

<REQUIRES_CUSTOM_SEGMENT_EXPRESSION>

<REQUIRES_CUSTOM_SEGMENT_TOKEN>

order to be able to successfully generate code from the template.

Allows you to specify that a particular custom field selection loop expression is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom field selection loop token is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom

<REQUIRES_CUSTOM_SELECTION_EXPRESSION>

<REQUIRES_CUSTOM_SELECTION_TOKEN>

<REQUIRES_CUSTOM_STRUCTLOOP_EXPRESSION>

structure loop expression is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_STRUCTLOOP_TOKEN>

Allows you to specify that a particular custom structure token is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_TAG_EXPRESSION>

Allows you to specify that a particular custom tag loop expression is required in order to be able to successfully generate code from the template.

<REQUIRES_CUSTOM_TAG_TOKEN>

Allows you to specify that a particular custom tag loop token is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom generic or structure expression is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular custom generic or structure token is required in order to be able to successfully

<REQUIRES_CUSTOM_EXPRESSION>

<REQUIRES_CUSTOM_TOKEN>

generate code from the template.

Allows you to specify that a particular command line option is required in order to be able to successfully generate code from the template.

Allows you to specify that a particular user-defined token is required in order to be able to successfully generate code from the template.

<REQUIRES_OPTION>

<REQUIRES_USERTOKEN>

CODEGEN FILENAME

This file header token allows you to define the name of the output file from within a template file. The name of the output file should appear between the tokens opening and closing tags. The opening and closing tags must be on the same line.

File header tokens must be grouped together at the top of the template file.

Usage

<CODEGEN_FILENAME>SomeFile.txt</CODEGEN_FILENAME>

The file name that you specify may contain generic expansion tokens or structure expansion tokens, which will be expanded as normal. For example:

<CODEGEN_FILENAME>Get<StructureName>.dbl</CODEGEN_FILENAM

Alternate forms

None

See also

• Other file header tokens

Video

Here's the Controlling Output File Names video. Make sure you select a high-definition version of the video, by default YouTube tends to play the lowest resolution which can look pretty awful!

CODEGEN FOLDER

This file header token allows you to specify that any output files created from the template should be placed into a named sub-folder below the main output folder. The name of the sub-folder should appear between the tokens opening and closing tags. The opening and closing tags must be on the same line. File header tokens must be grouped together at the top of the template file.

To enable the use of this token you must use the -tf command line option.

By default, if a namespace is being used, then the folder name will also appended to the namespace, but that behavior can be overridden by using the - nf command line option.

Usage

<CODEGEN FOLDER>ViewModel</CODEGEN FOLDER>

Alternate forms

None

See also

Other file header tokens

Copyright © 2018 Synergex International, Inc.

OPTIONAL_USERTOKEN

This file header token allows you to provide a default value for an optional user defined token. If the user-defined token has been specified via other mechanisms then that value will be used when generating code from the template, but if not then the default value specified in this token will be used.

File header tokens must be grouped together at the top of the template file.

Usage

<OPTIONAL_USERTOKEN>MYTOKEN=MyValue/OPTIONAL_USERTOI

Alternate forms

None

- <REQUIRES_USERTOKEN>
- User defined expansion tokens
- Other file header tokens

PROCESS TEMPLATE

This file header token allows you to specify that an additional template is to be processed after the processing of the current template completes. The name of the additional template to process is specified between the opening and closing tags to the token.

File header tokens must be grouped together at the top of the template file.

Usage

<PROCESS_TEMPLATE>other_template.tpl</PROCESS_TEMPLATE>

Alternate forms

None

See also

- <PROVIDE_FILE>
- Other file header tokens

Video

Here's the Configuring Template Dependencies video. Make sure you select a high-definition version of the video, by default YouTube tends to play the lowest resolution which can look pretty awful!

PROVIDE FILE

This file header token allows you to specify that an additional file is also provided to the output directory as part of the processing of the current template. The name of the file to be provided can include a logical name, otherwise CodeGen will expect to find the file in the same directory as your template files. If the file already exists in the output directory then it will not be replaced, even if the -r command line option is specified.

File header tokens must be grouped together at the top of the template file.

Usage

<PROVIDE_FILE>somefile.def</PROVIDE_FILE>
<PROVIDE_FILE>INC:somefile.def</PROVIDE_FILE>

Alternate forms

None

See also

- <PROCESS_TEMPLATE>
- Other file header tokens

Copyright © 2018 Synergex International, Inc.

REQUIRES CODEGEN VERSION

This file header token allows you to specify that a minimum version of CodeGen is required in order to successfully process the template. If the version of CodeGen being used does not satisfy this requirement then processing will terminate and an error message will be displayed.

NOTE: This token was introduced in CodeGen 5.2.1 and earlier versions will not process this token correctly. If a template containing this token is processed by an earlier version of CodeGen then it will not be recognized as a valid token and the token will simply be propagated into the output stream. This will almost certainly cause the generated code to fail, and the presence of the token in your output file will be an indication of the problem.

File header tokens must be grouped together at the top of the template file.

Usage

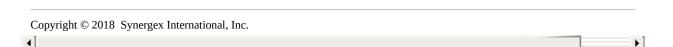
<REQUIRES_CODEGEN_VERSION>5.2.1</REQUIRES_CODEGEN_VERS

Alternate forms

None

See also

Other file header tokens



REQUIRES_CUSTOM_BUTTON_EXPRESSION

This file header token allows you to specify that a particular custom button loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

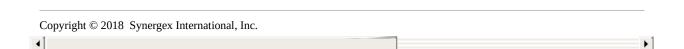
Usage

<REQUIRES_CUSTOM_BUTTON_EXPRESSION>MYEXPRESSION</REC

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES CUSTOM BUTTON TOKEN

This file header token allows you to specify that a particular custom button loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_BUTTON_TOKEN>MYTOKEN</REQUIRES_CUS

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES_CUSTOM_ENUM_EXPRESSION

This file header token allows you to specify that a particular custom enumeration loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_ENUM_EXPRESSION>MYEXPRESSION</REQUI

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES CUSTOM ENUM TOKEN

This file header token allows you to specify that a particular custom enum loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_ENUM_TOKEN>MYTOKEN</REQUIRES_CUST(

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES CUSTOM ENUM MEMBER EXPRES

This file header token allows you to specify that a particular custom enumeration member loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_ENUM_MEMBER_EXPRESSION>MYEXPRESSI(

Alternate forms

None

See also

- Custom Expression Tokens
- Other file header tokens

REQUIRES CUSTOM ENUM MEMBER TOKEN

This file header token allows you to specify that a particular custom enum member loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_ENUM_MEMBER_TOKEN>MYTOKEN</REQUIR

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES CUSTOM EXPRESSION

This file header token allows you to specify that a particular custom generic or structure expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_EXPRESSION>MYEXPRESSION</REQUIRES_CU

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES_CUSTOM_FIELD_EXPRESSION

This file header token allows you to specify that a particular custom field loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_FIELD_EXPRESSION>MYEXPRESSION</REQUI

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES_CUSTOM_FIELD_TOKEN

This file header token allows you to specify that a particular custom field loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_FIELD_TOKEN>MYTOKEN</REQUIRES_CUSTO

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES_CUSTOM_FILE_EXPRESSION

This file header token allows you to specify that a particular custom file loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_FILE_EXPRESSION>MYEXPRESSION</REQUIR

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES CUSTOM FILE TOKEN

This file header token allows you to specify that a particular custom file loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_FILE_TOKEN>MYTOKEN</REQUIRES_CUSTON

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES CUSTOM KEY EXPRESSION

This file header token allows you to specify that a particular custom key loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_KEY_EXPRESSION>MYEXPRESSION</REQUIR

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES CUSTOM KEY TOKEN

This file header token allows you to specify that a particular custom key loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

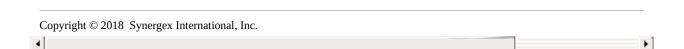
Usage

<REQUIRES_CUSTOM_KEY_TOKEN>MYTOKEN</REQUIRES_CUSTON

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES_CUSTOM_LOOPUTIL_EXPRESSION

This file header token allows you to specify that a particular custom loop utility expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_LOOPUTIL_EXPRESSION>MYEXPRESSION</RI

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES_CUSTOM_LOOPUTIL_TOKEN

This file header token allows you to specify that a particular custom loop utility token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_LOOPUTIL_TOKEN>MYTOKEN</REQUIRES_CI

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES_CUSTOM_NOTINLOOP_EXPRESSIO

This file header token allows you to specify that a particular custom "not in loop" expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_NOTINLOOP_EXPRESSION>MYEXPRESSION</

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES_CUSTOM_NOTINLOOP_TOKEN

This file header token allows you to specify that a particular custom "not in loop" token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_NOTINLOOP_TOKEN>MYTOKEN</REQUIRES_

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES CUSTOM RELATION EXPRESSION

This file header token allows you to specify that a particular custom relation loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_RELATION_EXPRESSION>MYEXPRESSION</RI

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES CUSTOM RELATION TOKEN

This file header token allows you to specify that a particular custom relation loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_RELATION_TOKEN>MYTOKEN</REQUIRES_CU

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES CUSTOM SEGMENT EXPRESSION

This file header token allows you to specify that a particular custom key segment loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_SEGMENT_EXPRESSION>MYEXPRESSION</RE</pre>

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES_CUSTOM_SEGMENT_TOKEN

This file header token allows you to specify that a particular custom key segment loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_SEGMENT_TOKEN>MYTOKEN</REQUIRES_CU

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES CUSTOM SELECTION EXPRESSION

This file header token allows you to specify that a particular custom field selection loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_SELECTION_EXPRESSION>MYEXPRESSION</F</pre>

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES_CUSTOM_SELECTION_TOKEN

This file header token allows you to specify that a particular custom field selection loop loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_SELECTION_TOKEN>MYTOKEN</REQUIRES_(

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES CUSTOM STRUCTLOOP EXPRESSI

This file header token allows you to specify that a particular custom structure loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_STRUCTLOOP_EXPRESSION>MYEXPRESSION

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES_CUSTOM_STRUCTLOOP_TOKEN

This file header token allows you to specify that a particular custom structure loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_STRUCTLOOP_TOKEN>MYTOKEN</REQUIRES

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES CUSTOM TAG EXPRESSION

This file header token allows you to specify that a particular custom tag loop expression is required in order to be able to successfully generate code from the template. If the specified custom expression has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_TAG_EXPRESSION>MYEXPRESSION</REQUIRED

Alternate forms

None

- Custom Expression Tokens
- Other file header tokens



REQUIRES_CUSTOM_TAG_TOKEN

This file header token allows you to specify that a particular custom tag loop token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

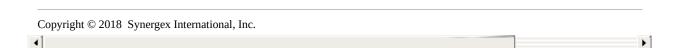
Usage

<REQUIRES_CUSTOM_TAG_TOKEN>MYTOKEN</REQUIRES_CUSTON

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES_CUSTOM_TOKEN

This file header token allows you to specify that a particular custom generic or structure expansion token is required in order to be able to successfully generate code from the template. If the specified custom token has not been provided when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_CUSTOM_TOKEN>MYTOKEN</REQUIRES_CUSTOM_TOI

Alternate forms

None

- Custom Expansion Tokens
- Other file header tokens



REQUIRES_OPTION

This file header token allows you to specify that a particular command line option is required in order to be able to successfully generate code from the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_OPTION>code</REQUIRES_OPTION>

Supported Options

Code	Command Line Option	Description
FL	-f l	Include fields which are "Excluded by Language" in field loop processing.
FO	-f o	Include overlay fields in field loop processing.
FR	-f r	Exclude fields which are "Excluded by ReportWriter" from field loop processing.
FT	-f t	Exclude fields which are "Excluded by Tookit" from field loop processing.
FW	-f w	Exclude fields which are "Excluded by Web" from field loop processing.
PREFIX	-prefix <value></value>	Prefix field names with <value>.</value>
SUBSET	-subset <name></name>	Only process fields belonging to the named subset.
TF	-tf	Allow template-defined folders.

If the required command line option was not used when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

Alternate forms

None

See also

- Command Line Reference
- Other file header tokens

REQUIRES_USERTOKEN

This file header token allows you to specify that a particular user-defined token is required in order to be able to successfully generate code from the template. If the required user-defined token has not been specified when attempting to generate code from the template then CodeGen will display an error and will stop processing the template.

File header tokens must be grouped together at the top of the template file.

Usage

<REQUIRES_USERTOKEN>MYTOKEN</REQUIRES_USERTOKEN>

Alternate forms

None

See also

- <OPTIONAL_USERTOKEN>
- User defined expansion tokens
- Other file header tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

Generic Tokens and Expressions

Generic tokens are tokens that do not rely on any specific source of repository metadata, and which can therefor be used at anywhere in a template file.

- Generic expansion tokens
- Generic expression tokens

Generic Expansion Tokens

Generic expansion tokens are tokens which insert generic (non structure specific) information or take some generic action, and can be used at anywhere in a template file.

Generic Token	Description
<author></author>	Inserts your name, which you specify via the environment variable CODEGEN_AUTHOR.
<codegen_version></codegen_version>	Inserts a CodeGen version identifier.
<company></company>	Inserts your company name, which you specify via the environment variable CODEGEN_COMPANY.
<counter_1_decrement></counter_1_decrement>	Decrements the value of counter 1 by one.
<counter_1_increment></counter_1_increment>	Increments the value of counter 1 by one.
<counter_1_reset></counter_1_reset>	Resets the value of counter 1 to zero.
<counter_1_value></counter_1_value>	Inserts the current value of counter 1 into the code stream.
<counter_2_decrement></counter_2_decrement>	Decrements the value of counter 2 by one.
<counter_2_increment></counter_2_increment>	Increments the value of counter 2 by one.
<counter_2_reset></counter_2_reset>	Resets the value of counter 2 to zero.
<counter_2_value></counter_2_value>	Inserts the current value of counter 2 into the code stream.
<database></database>	Inserts the name of the current relational database being targeted. For more information refer to the command line

<date></date>	option -database. Inserts the current date in MM/DD/YYYY format.
<date1></date1>	Inserts the current date in DD/MM/YYYY format.
<day></day>	Inserts the current numeric day of the month in DD format
<env:variable></env:variable>	Inserts the value of an environment variable into the output stream. If the environment variable does not exist then an error is generated.
<envifexist:variable></envifexist:variable>	Inserts the value of an environment variable into the output stream. If the environment variable does not exist then the token will be ignored and no error will be generated.
<file:name.ext></file:name.ext>	Inserts the contents of a file into the output stream. The file specification can include an environment variable. If the file does not exist then an error is generated.
<fileifexist:name.ext></fileifexist:name.ext>	Inserts the contents of a file into the output stream. The file specification can include an environment variable. If the file does not exist then the token is ignored and no error is generated.
<guid1>, <guid2>, <guid3></guid3></guid2></guid1>	Inserts one of three GUID values that are unique to the current CodeGen task.
<host_dns_name></host_dns_name>	Inserts the DNS name of the computer that you are running CodeGen on. Inserts the IP address name of the computer that you are running CodeGen on. Inserts the current numeric month number in MM format.
<host_ip_address></host_ip_address>	
<month></month>	

<monthname></monthname>	Inserts the alphanumeric name of the current month.
<monthshortname></monthshortname>	Inserts the abbreviated alphanumeric name of the current month.
<namespace></namespace>	Inserts the default namespace which can be specified either via the environment variable SYNDEFNS, or via the -n command line option.
<template></template>	Inserts the base name of the template file. The base name of the template is the name of the template file with no file extension.
<random_10></random_10>	Inserts a random number in the range of 1 to 10.
<random_100></random_100>	Inserts a random number in the range of 1 to 100.
<random_1000></random_1000>	Inserts a random number in the range of 1 to 1000.
<random_int></random_int>	Inserts a random number in the range of 1 to 2147483647.
<time></time>	Inserts the current time in 24 hour HH:MM format.
<weekday></weekday>	Inserts the name of the current weekday.
<year></year>	Inserts the current year in YYYY format.

AUTHOR

Inserts your name, which you specify via the environment variable CODEGEN AUTHOR.

When you install CodeGen you are prompted to enter your name during the installation process and the value that you enter is saved in the Windows Registry. If CodeGen finds that the environment variable CODEGEN_AUTHOR is not set then it will attempt to read this value from the registry and will set the CODEGEN_AUTHOR environment variable automatically.

If no author name can be determined the string **CodeGen** is inserted.

Usage

<AUTHOR>

Alternate forms

None

See also

• Other generic tokens

Video

Here's the Using AUTHOR and COMPANY Tokens video. Make sure you select a high-definition version of the video, by default YouTube tends to play the lowest resolution which can look pretty awful!

CODEGEN_VERSION

Inserts a CodeGen version identifier. For example CodeGen V5.0.0

Usage

<CODEGEN_VERSION>

Alternate forms

None

See also

• Other generic tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

COMPANY

This token is inserts your company name, which you specify via the environment variable CODEGEN_COMPANY. For example **Synergex Professional Services Group**

When you install CodeGen on a Windows system you are prompted to enter your company during the installation process and the value that you enter is saved in the Windows Registry. If CodeGen finds that the environment variable CODEGEN_COMPANY is not set then it will attempt to read this value from the registry and will set the CODEGEN_COMPANY environment variable automatically.

If no company name can be determined the string **Synergex** is inserted.

Usage

<COMPANY>

Alternate forms

None

See also

• Other generic tokens

Video

Here's the Using AUTHOR and COMPANY Tokens video. Make sure you select a high-definition version of the video, by default YouTube tends to play the lowest resolution which can look pretty awful!

COUNTER 1 DECREMENT

CodeGen provides two built in counters that developers can use when designing template files. For each code generation task both counters start off with a value of zero. Tokens in the template file can cause the value of each counter to be incremented, decremented or reset to zero. Another token can be used to insert the current value of a counter into the text output stream.

This token decrements the value of counter 1 by 1. If the token appears inside of an expression that currently evaluates to false then the value will not be decremented.

This token does not cause any text to be written to the output stream.

Usage

<COUNTER 1 DECREMENT>

Alternate forms

None

See also

Other generic tokens

COUNTER 1 INCREMENT

CodeGen provides two built in counters that developers can use when designing template files. For each code generation task both counters start off with a value of zero. Tokens in the template file can cause the value of each counter to be incremented, decremented or reset to zero. Another token can be used to insert the current value of a counter into the text output stream.

This token increments the value of counter 1 by 1. If the token appears inside of an expression that currently evaluates to false then the value will not be incremented.

This token does not cause any text to be written to the output stream.

Usage

<COUNTER 1 INCREMENT>

Alternate forms

None

See also

• Other generic tokens

COUNTER 1 RESET

CodeGen provides two built in counters that developers can use when designing template files. For each code generation task both counters start off with a value of zero. Tokens in the template file can cause the value of each counter to be incremented, decremented or reset to zero. Another token can be used to insert the current value of a counter into the text output stream.

This token resets the value of counter 1 to 0. If the token appears inside of an expression that currently evaluates to false then the value will not be reset.

This token does not cause any text to be written to the output stream.

Usage

<COUNTER_1_RESET>

Alternate forms

None

See also

Other generic tokens

COUNTER_1_VALUE

CodeGen provides two built in counters that developers can use when designing template files. For each code generation task both counters start off with a value of zero. Tokens in the template file can cause the value of each counter to be incremented, decremented or reset to zero. Another token can be used to insert the current value of a counter into the text output stream.

Inserts the current value of counter 1 into the output stream. If the token appears inside of an expression that currently evaluates to false then the value will not be inserted.

Usage

<COUNTER_1_VALUE>

Alternate forms

None

See also

• Other generic tokens

COUNTER 2 DECREMENT

CodeGen provides two built in counters that developers can use when designing template files. For each code generation task both counters start off with a value of zero. Tokens in the template file can cause the value of each counter to be incremented, decremented or reset to zero. Another token can be used to insert the current value of a counter into the text output stream.

This token decrements the value of counter 2 by 1. If the token appears inside of an expression that currently evaluates to false then the value will not be decremented.

This token does not cause any text to be written to the output stream.

Usage

<COUNTER 2 DECREMENT>

Alternate forms

None

See also

Other generic tokens

COUNTER 2 INCREMENT

CodeGen provides two built in counters that developers can use when designing template files. For each code generation task both counters start off with a value of zero. Tokens in the template file can cause the value of each counter to be incremented, decremented or reset to zero. Another token can be used to insert the current value of a counter into the text output stream.

This token increments the value of counter 2 by 1. If the token appears inside of an expression that currently evaluates to false then the value will not be incremented.

This token does not cause any text to be written to the output stream.

Usage

<COUNTER_2_INCREMENT>

Alternate forms

None

See also

• Other generic tokens

COUNTER 2 RESET

CodeGen provides two built in counters that developers can use when designing template files. For each code generation task both counters start off with a value of zero. Tokens in the template file can cause the value of each counter to be incremented, decremented or reset to zero. Another token can be used to insert the current value of a counter into the text output stream.

This token resets the value of counter 2 to 0. If the token appears inside of an expression that currently evaluates to false then the value will not be reset.

This token does not cause any text to be written to the output stream.

Usage

<COUNTER_2_RESET>

Alternate forms

None

See also

Other generic tokens

COUNTER_2_VALUE

CodeGen provides two built in counters that developers can use when designing template files. For each code generation task both counters start off with a value of zero. Tokens in the template file can cause the value of each counter to be incremented, decremented or reset to zero. Another token can be used to insert the current value of a counter into the text output stream.

Inserts the current value of counter 2 into the output stream. If the token appears inside of an expression that currently evaluates to false then the value will not be inserted.

Usage

<COUNTER 2 VALUE>

Alternate forms

None

See also

• Other generic tokens

DATABASE

Inserts the name of the current relational database being targeted.

Usage

<DATABASE>

Alternate forms

None

Possible Values

- MySQL
- PostgreSQL
- SQLServer

See also

- Command line option -database
- Environment variable CODEGEN_DATABASE_TYPE

DATE

Inserts the current date in MM/DD/YYYY format. For example 10/23/2008

Usage

<DATE>

Alternate forms

None

See also

• Other generic tokens

DATE1

Inserts the current date in DD/MM/YYYY format. For example 23/10/2008

Usage

<DATE1>

Alternate forms

None

See also

• Other generic tokens

DAY

Inserts the current numeric day of the month in DD format. For example 23

Usage

<DAY>

Alternate forms

None

See also

• Other generic tokens

ENV:variable

Inserts the value of an environment variable into the output stream. If the environment variable does not exist then an error is generated.

Note that the translation value of the environment variable may generic and structure expansion tokens.

Usage

<ENV:variable>

Alternate forms

None

See also

• Other generic tokens

ENVIFEXIST: variable

Inserts the value of an environment variable into the output stream. If the environment variable does not exist then the token will be ignored and no error will be generated.

Note that the translation value of the environment variable may generic and structure expansion tokens.

Usage

<ENVIFEXIST:variable>

Alternate forms

None

See also

• Other generic tokens

FILE:name.ext

This token reads the content of a text file and inserts the resulting data into the output stream. The file can either be located in the current directory, or you can include a logical name specification to specify the location of the file. If the file is not found then an error is generated.

The file may contain any number of lines of text, and may also contain generic or structure expansion tokens, which will be expanded in the usual way. If the token exists on the same line as and subsequent to other text, then the first record from the file will follow that text in the resulting output file. Any subsequent records from the file will appear on new lines in the output file.

Usage

<FILE:name.ext>

<FILE:LOCATION:name.ext>

Alternate forms

None

See also

• Other generic tokens

Video

Here's the Using Template Include Files video. Make sure you select a high-definition version of the video, by default YouTube tends to play the lowest resolution which can look pretty awful!

FILEIFEXIST:name.ext

This token reads the content of a text file and inserts the resulting data into the output stream. The file can either be located in the current directory, or you can include a logical name specification to specify the location of the file. If the file is not found then the token is ignored and no error is generated.

The file may contain any number of lines of text, and may also contain generic or structure expansion tokens, which will be expanded in the usual way. If the token exists on the same line as and subsequent to other text, then the first record from the file will follow that text in the resulting output file. Any subsequent records from the file will appear on new lines in the output file.

Usage

<FILEIFEXIST:name.ext>

<FILEIFEXIST:LOCATION:name.ext>

Alternate forms

None

See also

• Other generic tokens

Video

Here's the Using Template Include Files video. Make sure you select a high-definition version of the video, by default YouTube tends to play the lowest resolution which can look pretty awful!

GUID1, GUID2, GUID3

CodeGen supports inserting one of up to three GUID values (e.g. {8d881f51-8d3e-4ee7-9e20-17ee24452da9}) into the output stream.

Each time you run CodeGen it will generate new values for the three GUID values exposed by these tokens. These GUID values will only be consistent during a single invocation of CodeGen. The next time you invoke CodeGen new values will be generated.

If you need to use the same GUID value in multiple templates then you must process those templates all at the same time during a single invocation of CodeGen, or alternatively you could generate your own GUID values and inject them via a user-defined token.

Usage

<GUID1>

<GUID2>

<GUID3>

Alternate forms

None

See also

Other generic tokens

HOST_DNS_NAME

Inserts the DNS name of the computer that you are running CodeGen on. If there is a problem determining the DNS name then "LOCALHOST" is used.

Usage

<HOST_DNS_NAME>

Alternate forms

<host_dns_name>

See also

• Other generic tokens

HOST_IP_ADDRESS

Inserts the IP address name of the computer that you are running CodeGen on. If the system has multiple IP addresses then the first IP address reported by Windows is used. If there is a problem determining the IP address then the "localhost" address of "127.0.0.1" is used.

Usage

<HOST_IP_ADDRESS>

Alternate forms

None

See also

Other generic tokens

MONTH

Inserts the current numeric month (MM). For example: 10

Usage

<MONTH>

Alternate forms

None

See also

• Other generic tokens

MONTHNAME

Inserts the alphanumeric name of the current month. For example: October

Usage

<MONTHNAME>

Alternate forms

None

See also

• Other generic tokens

MONTHSHORTNAME

Inserts the abbreviated alphanumeric name of the current month. For example: Oct

Usage

<MONTHSHORTNAME>

Alternate forms

None

See also

• Other generic tokens

NAMESPACE

Inserts the default namespace, which can be specified either via the environment variable SYNDEFNS, or via the -n command line option. For example: SynPSG.CodeGen

If a namespace token is encountered and no default namespace can be determined then CodeGen will report an error.

Usage

<NAMESPACE>

Alternate forms

None

See also

• Other generic tokens

RANDOM_10

Inserts a random number in the range of 1 to 10.

Usage

<RANDOM_10>

Alternate forms

None

See also

• Other generic tokens

RANDOM_100

Inserts a random number in the range of 1 to 100.

Usage

<RANDOM_100>

Alternate forms

None

See also

• Other generic tokens

RANDOM_1000

Inserts a random number in the range of 1 to 1000.

Usage

<RANDOM_100>

Alternate forms

None

See also

• Other generic tokens

RANDOM_INT

Inserts a random number in the range of 1 to 2147483647.

Usage

<RANDOM_INT>

Alternate forms

None

See also

• Other generic tokens

TEMPLATE

Inserts the base name of the template file. The base name of the template is the name of the template file with no file extension. For example, if processing a template called get_record.tpl: GET_RECORD

Usage

<TEMPLATE>

Alternate forms

<template>

See also

• Other generic tokens

TIME

<TIME>

Inserts the current time in 24 hour HH:MM format. For example: 16:06

Usage

Alternate forms

None

See also

• Other generic tokens

WEEKDAY

Inserts the name of the current weekday. For example: Thursday

Usage

<WEEKDAY>

Alternate forms

None

See also

• Other generic tokens

YEAR

Inserts the current year in YYYY format. For example 2010

Usage

<YEAR>

Alternate forms

None

See also

• Other generic tokens

Generic Expression Tokens

Generic expressions are expressions that are valid for use anywhere in a template file.

Generic Expression	Description
<if counter_n=""></if>	The current value of counter n is non-zero. The value of n must be 1 or 2.
	Evaluates the current value of counter n based on the specified operator and value. The value of n must be 1 or 2, corresponding to one of CodeGen's two template based counters, and the comparison value val must be an integer literal. Possible values for op are:
<if counter_n_op_val=""></if>	EQ Equal to NE Not equal to LT Less than GT Greater than LE Less than or equal to GE Greater than or equal to DB Divisible by
	An example of using this expression is:
	<if counter_1_le_10=""> template code for the first 10 items <else> template code for subsequent items </else></if>
<if database_mysql=""></if>	The database being targeted is MySQL. Refer to command line option -database.
<if DATABASE_POSTGRESQL2</if 	The database being targeted is PostgreSOI
<if< td=""><td>The database being targeted is SQL Server.</td></if<>	The database being targeted is SQL Server.

DATABASE_SQLSERVER>	Refer to command line option -database.
<if debug_logging=""></if>	CodeGen is being run with debug logging enabled (-d command line option).
<if field_prefix=""></if>	A field prefix string has been specified (- prefix command line option)
<if field_subset=""></if>	CodeGen is processing a subset of fields (-subset command line option or -fields command line option).
<if MULTIPLE_STRUCTURES></if 	Multiple structures are being processed at the same time (-ms command line option).
<if namespace=""></if>	A namespace has been specified (-n command line option).
<if not_counter_1=""></if>	The current value of counter 1 is zero.
<pre><if not_counter_2=""></if></pre>	The current value of counter 2 is zero.
<if verbose_logging=""></if>	CodeGen is being run with verbose logging enabled (-v command line option).

Structure Tokens and Expressions

Structure tokens are tokens that are associated with a repository structure that is currently being processed, and can be used anywhere in a template file when a repository structure is being processed. There are two types of structure token:

- Structure expansion tokens
- Structure expression tokens

Structure Expansion Tokens

Structure expansion tokens are expansion tokens that are used to insert information about the current Repository structure into the output stream. They can be used anywhere in a template file.

Structure Token	Description
<data_fields_list></data_fields_list>	Inserts a comma-separated list of a structures data fields (i.e. non primary key) fields.
<display_field></display_field>	Inserts the name of the structures display field, if defined.
<file_addressing></file_addressing>	Inserts the addressing size of the structures first assigned data file.
<file_compression></file_compression>	Inserts a string indicating whether the structures first assigned file has data compression enabled.
<file_density></file_density>	Inserts the default key density of the structures first assigned file.
<file_desc></file_desc>	Inserts the description of the first data file assigned to the structure.
<file_name></file_name>	Inserts the name of the first data file assigned to the structure.
<file_name_noext></file_name_noext>	Inserts the name of the first data file assigned to the structure, with the file extension removed.
<file_pagesize></file_pagesize>	Inserts the page size of the structures first assigned file.
<pre><file_portable_int_specs></file_portable_int_specs></pre>	Inserts the portable integer positions

<file_rectype></file_rectype>	string of the first data file assigned to the current structure. Inserts a string indicating the record type of the first data file assigned to the structure.
<file_rps_name></file_rps_name>	Inserts the repository name of the first file definition that is assigned to the structure being processed. Inserts a string indicating whether the
<file_static_rfa></file_static_rfa>	structures first assigned file has static RFA's enabled.
<file_type></file_type>	Inserts a string indicating the type of the first data file assigned to the structure.
<file_utext></file_utext>	Inserts the user text string of the first data file assigned to the structure.
<mapped_file></mapped_file>	Inserts the file specification of the first file assigned to the mapped structure.
<mapped_structure></mapped_structure>	Inserts the name of the mapped structure.
<primary_key_field></primary_key_field>	Inserts the name of the first segment of the structures primary key. This token only works with structures with single segment primary keys.
<structure_#n></structure_#n>	Switches context to a different structure. Only valid when processing multiple structures with the -ms option.
<structure_children></structure_children>	Inserts the number of the structures first level child members (fields and groups).
<structure_desc></structure_desc>	Inserts the structure description.
	Inserts the total number of fields in the

<structure_fields></structure_fields>	structure. If the structure includes any array fields then the individual array field elements are counted towards the number of fields in the structure.
<structure_keys></structure_keys>	Inserts the total number of keys defined in the structure.
<structure_ldesc></structure_ldesc>	Inserts the structures long description.
<structure_name></structure_name>	Inserts the name of the Repository structure that is currently being processed, or the structures alias name if specified.
<structure_noalias></structure_noalias>	Inserts the name of the Repository structure that is currently being processed. This token is always replaced by the actual name of the repository structure, even if an alias name has been specified.
<structure_noplural></structure_noplural>	Inserts the name of the structure that is currently being processed. If the name ends with an "S" character then it will be removed.
<structure_plural></structure_plural>	Inserts the name of the structure that is currently being processed. If the name does not end with an "S" character then one will be added.
<structure_size></structure_size>	Inserts the structure size in bytes.
<structure_utext></structure_utext>	Inserts the structure user text string.

DATA_FIELDS_LIST

Inserts a comma-separated list of a structures data fields (i.e. non primary key) fields. This is mainly useful for defining sets in UI Toolkit input windows, as any field marked "Excluded by Toolkit" will NOT be included in the list of fields. Array fields are listed separately with real a real array index.

Usage

<DATA_FIELDS_LIST>

Example Output

FIELD1,FIELD2,FIELD3[1],FIELD3[2],FIELD4

Alternate forms

None

See also

• Other structure tokens

DISPLAY_FIELD

Inserts the name of the structures display field, if defined. A structures display field is defined by adding the text **@CODEGEN_DISPLAY_FIELD** to the fields user defined text string or long description.

CodeGen will display an error if it encounters this token and the structure being processed does not have a display field defined.

Usage

<DISPLAY_FIELD>

Example Output

DEPARTMENT_NAME

Alternate forms

None

See also

Other structure tokens

FILE_ADDRESSING

Inserts the addressing size of the structures first assigned data file.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_ADDRESSING>

Possible values

- 32
- 40

Alternate forms

None

See also

• Other structure tokens

FILE_CHANGE_TRACKING

Inserts a string indicating whether the structures first assigned file has change tracking enabled.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

This option is supported by Synergy/DE V10.1 and higher.

Usage

<FILE_CHANGE_TRACKING>

Possible values

- YES
- NO

Alternate forms

<file_change_tracking>

See also

• Other structure tokens

FILE_COMPRESSION

Inserts a string indicating whether the structures first assigned file has data compression enabled.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_COMPRESSION>

Possible values

- YES
- NO

Alternate forms

<file_compression>

See also

• Other structure tokens

FILE_DENSITY

Inserts the default key density of the structures first assigned file.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_DENSITY>

Possible values

- 50
- 100

Alternate forms

None

See also

Other structure tokens

FILE_DESC

Inserts the description of the first data file assigned to the structure. For example:

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_DESC>

Example Output

Customer Master File

Alternate forms

None

See also

Other structure tokens

FILE_NAME

Inserts the name of the first data file assigned to the current structure.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_NAME>

Example Output

DAT:customer.ism

Alternate forms

None

See also

Other structure tokens

FILE_NAME_NOEXT

Inserts the name of the first data file assigned to the current structure, with the file extension removed. This token can be useful when constructing file specifications where the file extension is based on a variable company code or other data.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_NAME_NOEXT>

Example Output

DAT:customer

Alternate forms

None

See also

Other structure tokens

FILE_ODBC_NAME

Inserts the ODBC table name of the first repository file definition that is assigned to the structure currently being processed. If no ODBC table name is specified then the name of the first assigned structure is used.

Usage

<FILE_ODBC_NAME>

Example Output

CUSTOMER

Alternate forms

- <file_odbc_name>
- <File_Odbc_Name>
- <File_odbc_name>
- <FileOdbcName>
- <fileOdbcName>

See also

FILE PAGESIZE

Inserts the page size of the current structures first assigned file.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_PAGESIZE>

Possible values

- 512
- 1024
- 2048
- 4096
- 8192
- 16384
- 32768

Note: Page sizes 16384 and 32768 were introduced with ISAM Rev 6 in Synergy/DE 10.1

Alternate forms

None

See also

FILE_PORTABLE_INT_SPECS

Inserts the portable integer positions string of the first data file assigned to the current structure.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_PORTABLE_INT_SPECS>

Example Output

Alternate forms

None

See also

• Other structure tokens

FIELD_PREFIX

Inserts the current field prefix that was specified via the -prefix command line option. If the -prefix command line option was not used then this the token does not write anything to the output file.

Usage

<FIELD_PREFIX>

Alternate forms

None

See also

• Other structure tokens

FILE_RECTYPE

Inserts a string indicating the record type of the first data file assigned to the structure.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_RECTYPE>

Possible values

- FIXED
- VARIABLE
- MULTIPLE

Alternate forms

<file_rectype>

See also

Other structure tokens

FILE_RPS_NAME

Inserts the name of the first repository file definition that is assigned to the structure currently being processed.

Usage

<FILE_RPS_NAME>

Example Output

CUSTOMER

Alternate forms

- <file_rps_name>
- <File_Rps_Name>
- <File_rps_name>
- <FileRpsName>
- <fileRpsName>

See also

• Other structure tokens

FILE_STATIC_RFA

Inserts a string indicating whether the structures first assigned file has static RFA's enabled.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_STATIC_RFA>

Possible values

- YES
- NO

Alternate forms

<file_static_rfa>

See also

Other structure tokens

FILE_STORED_GRFA

Inserts a string indicating whether the structures first assigned file has stored GRFA's enabled.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Supported in Synergy/DE V10.1 and higher.

Usage

<FILE_STORED_GRFA>

Possible values

- YES
- NO

Alternate forms

<file_stored_grfa>

See also

• Other structure tokens

FILE_TYPE

Inserts a string indicating the type of the first data file assigned to the structure.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_TYPE>

Possible values

- ASCII
- DBL ISAM
- RELATIVE
- USER DEFINED

Alternate forms

None

See also

Other structure tokens

FILE UTEXT

Inserts the user text string of the first data file assigned to the structure.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

Usage

<FILE_UTEXT>

Alternate forms

None

See also

Other structure tokens

MAPPED FILE

Inserts the file specification of the first file assigned to the mapped structure.

To use this token you must be processing a structure which has been mapped to a second structure, and the second structure must have been assigned to a repository file definition. CodeGen will display an error if it encounters this token and these requirements have not been satisfied.

For information on structure and field mapping, refer to Preparing Your Repository.

Usage

<MAPPED_FILE>

Example Output

DAT:department.ism

Alternate forms

None

See also

MAPPED_STRUCTURE

Inserts the name of the mapped structure. To use this token you must be processing a structure which has been mapped to a second structure. CodeGen will display an error if it encounters this token and you are not processing a structure with a structure mapping.

For information on structure and field mapping, refer to Preparing Your Repository.

Usage

<MAPPED_STRUCTURE>

Example Output

DEPARTMENT

Alternate forms

<mapped_structure>

See also

Other structure tokens

MAPPED TAG FIELD

Inserts the name of the name of the first tag field in the mapped structure.

To use this token you must be processing a structure which has been mapped to a second structure, and that structure must have a tag. CodeGen will display an error if it encounters this token and you are not processing a structure with a structure mapping.

For information on structure and field mapping, refer to Preparing Your Repository.

Usage

<MAPPED_TAG_FIELD>

Example Output

DEPARTMENT_ID

Alternate forms

<mapped_tag_field>

See also

PRIMARY KEY FIELD

Inserts the name of the first segment of the structures primary key.

WARNING: This token does not support structures with segmented primary keys. You can achieve more flexible results by using a key segment loop within a primary key block. This technique is discussed later in this chapter. This token is present for compatibility with older template files only, and will be removed in a future version.

Usage

<PRIMARY_KEY_FIELD>

Example Output

CUSTOMER_ID

Alternate forms

- <primary_key_field>
- <Primary_key_field>
- <Primary_Key_Field>
- <PrimaryKeyField>
- <primaryKeyField>

See also

STRUCTURE#n

This special token switches context to a different structure, and can only be used when processing multiple structures concurrently. The n character represents the one-based index number of the structure to switch to, and must be a number between 1 and the number of structures being processed.

CodeGen can process multiple structures concurrently in one of two ways:

- Multiple structures can be specified on the command line, after the -s
 command line option, and the -ms command line option can be used to
 specify that all of the structures are to be processed together.
- The -file command line option can be used to use a repository file definition as the starting point for code generation, and that file definition can have multiple structures assigned.

This token does not cause any data to be written to the output stream.

For an example of processing multiple structures in a single template file, refer to the example template named xf_net_get_master_detail.tpl.

Usage

<STRUCTURE#n>

Alternate forms

None

See also

STRUCTURE_CHILDREN

Inserts the number of the structures first level child members (fields and groups).

Usage

<STRUCTURE_CHILDREN>

Example Output

15

Alternate forms

None

See also

• Other structure tokens

STRUCTURE_DESC

Inserts the current structures description.

Usage

<STRUCTURE_DESC>

Example Output

Customer Master Record

Alternate forms

None

See also

• Other structure tokens

STRUCTURE_FIELDS

Inserts the total number of fields in the structure. If the structure includes any array fields then the individual array field elements are counted towards the number of fields in the structure.

Usage

<STRUCTURE_FIELDS>

Example Output

25

Alternate forms

None

See also

• Other structure tokens

STRUCTURE_KEYS

Inserts the total number of keys defined in the structure.

Usage

<STRUCTURE_KEYS>

Example Output

3

Alternate forms

None

See also

• Other structure tokens

STRUCTURE_LDESC

Inserts the structures long description.

Usage

<STRUCTURE_LDESC>

Alternate forms

None

See also

• Other structure tokens

STRUCTURE_NAME

Inserts the name of the structure that is currently being processed. If the -a command line option is being used then inserts the structures alias as specified on the command line. If you are using structure aliases and want to insert the name of the actual structure (not the alias) then use the <STRUCTURE_NOALIAS> token.

Usage

<STRUCTURE_NAME>

Example Output

CUSTOMER

Alternate forms

- <structure_name>
- <Structure_Name>
- <Structure_name>
- <StructureName>
- <structureName>

See also

- <STRUCTURE NOALIAS>
- <STRUCTURE_NOPLURAL>
- <STRUCTURE_PLURAL>
- Other structure tokens

STRUCTURE_NOALIAS

Inserts the name of the Repository structure that is currently being processed. This token is always replaced by the actual name of the repository structure, even if the -a command line option is being used.

Usage

<STRUCTURE_NOALIAS>

Example Output

CUSTOMER

Alternate forms

- <structure_noalias>
- <Structure_Noalias>
- <Structure_noalias>
- <StructureNoAlias>
- <structureNoAlias>

See also

- <STRUCTURE_NAME>
- <STRUCTURE_NOPLURAL>
- <STRUCTURE_PLURAL>
- Other structure tokens

STRUCTURE_NOPLURAL

Inserts the name of the structure that is currently being processed. If the -a command line option is being used then inserts the structures alias as specified on the command line. If the name ends with an "S" character then it will be removed.

Usage

<STRUCTURE_NOPLURAL>

Example Output

For a structure named CUSTOMER**S** the token will insert CUSTOMER

Alternate forms

- <structure_noplural>
- <Structure_Noplural>
- <Structure_noplural>
- <StructureNoplural>
- <structureNoplural>

See also

- <STRUCTURE_NAME>
- <STRUCTURE_NOALIAS>
- <STRUCTURE_PLURAL>
- Other structure tokens

STRUCTURE_PLURAL

Inserts the name of the structure that is currently being processed. If the -a command line option is being used then inserts the structures alias as specified on the command line. If the name does not end with the character "S" then one will be added.

Usage

<STRUCTURE_PLURAL>

Example Output

For a structure named CUSTOMER the token will insert CUSTOMERS

Alternate forms

- <structure_plural>
- <Structure Plural>
- <Structure_plural>
- <StructurePlural>
- <structurePlural>

See also

- <STRUCTURE_NAME>
- <STRUCTURE_NOALIAS>
- <STRUCTURE_NOPLURAL>
- Other structure tokens

STRUCTURE_SIZE

Inserts the size of the current structure expressed in bytes.

Usage

<STRUCTURE_SIZE>

Example Output

495

Alternate forms

None

See also

• Other structure tokens

STRUCTURE_UTEXT

Inserts the current structures user text string.

Usage

<STRUCTURE_UTEXT>

Alternate forms

None

See also

• Other structure tokens

Structure Expression Tokens

Structure expression tokens are expression tokens that allow you to test various attributes of the structure that is currently being processed. They are valid for use anywhere in a template file, but only when a repository structure is being processed.

Structure Expression	Description
<pre><if file_change_tracking=""></if></pre>	The file associated with the structure has change tracking enabled.
<if file_compression=""></if>	The file associated with the structure has data compression enabled.
<pre><if file_portable_int_specs=""></if></pre>	The file associated with the structure has portable integer locations specified.
<if file_static_rfa=""></if>	The file associated with the structure uses static RFA.
<if file_stored_grfa=""></if>	The file associated with the structure uses stored GRFA.
<if file_terabyte=""></if>	The file associated with the structure is a "terabyte" file.
<if structure_ascii=""></if>	The structure being processed is associated with an ASCII sequential file.
<if structure_files=""></if>	The structure currently being processed is assigned to one or more file definitions.
<if STRUCTURE_HAS_UNIQUE_KEY></if 	The structure currently being processed has at least one unique key. You can access the first unique key by using a <unique_key> block.</unique_key>
<if STRUCTURE_HAS_UNIQUE_PK></if 	The structure currently being processed has a unique primary key.
<if structure_isam=""></if>	The structure being processed is associated with an ISAM file.

<if structure_keys=""></if>	The structure currently being processed has one or more keys defined.
<if structure_ldesc=""></if>	The structure currently being processed has text in its long description field.
<if structure_mapped=""></if>	The structure being processed is mapped to another structure.
<if structure_relations=""></if>	The structure currently being processed has one or more relations defined.
<if structure_relative=""></if>	The structure being processed is associated with a RELATIVE file.
<if structure_tags=""></if>	The structure currently being processed has one or more tags defined.
<if STRUCTURE_MULTIPLE_TAGS></if 	The structure currently being processed has more than one tag defined.
<if structure_single_tag=""></if>	The structure currently being processed has exactly one tag defined.
<if STRUCTURE_USER_DEFINED></if 	The structure being processed is associated with a user defined file.
<if structure_utext=""></if>	The structure currently being processed has text in its user defined text field.

Field Loops

A field loop is a template file construct which allows you to iterate through the collection of fields that CodeGen has information about. These field definitions can come from one of two places:

- If you are generating code based on information obtained form a repository structure, the field collection is based on the fields which are defined in that repository structure.
- If you are generating code based on the definition of a UI Toolkit input window (defined in a window script file), the field collection is determined by the fields defined in that input window. Of course these fields may in turn refer to a repository structure.
- If you are generating code based on information obtained from a Synergy Method Catalog, and you are currently processing a structure parameter, the field collection is based on the fields which are defined in the repository structure associated with the current parameter.

Field loops are delimited by a matching pair of **FIELD_LOOP**> and **</FIELD_LOOP**> tags that surround the template code to be inserted for each field. The code between the opening and closing tags of a field loop is repeated for each field in the structure being processed.

- Field Loop Expansion Tokens
- Field Loop Expression Tokens

Field loops may be defined within structure loops and parameter loops, but may not be declared within any other loop construct.

CodeGen processes array fields as multiple individual fields. Normally CodeGen does not process overlay fields, or fields marked "Excluded by Language".

There are two types of field loops, called multi-line field loops, and in-line field loops.

Multi-Line Field Loops

Multi-line field loops occur when the opening and closing field loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
[code]
<FIELD_LOOP>code
</FIELD_LOOP>
[code]
```

Multi-line field loops generate one or more lines of output code for each field that is processed.

In-Line Field Loops

In-line field loops exist when the opening and closing field loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
[code] <FIELD_LOOP> code </FIELD_LOOP> [code]
```

In-line field loops generate code into the current output line.

Field Loop Example 1 (Multi-Line)

The following template file code:

```
sql = "CREATE TABLE <STRUCTURE_NAME> ("
     <FIELD_LOOP>
& + "<FIELD_SQLNAME> <FIELD_SQLTYPE><IF REQUIRED>
NOT NULL</IF><,>"
     </FIELD_LOOP>
& + ")"
```

could produce output like this:

```
sql = "CREATE TABLE PROJECT ("
& + "SYNERGYGRFA VARCHAR(22),"
& + "PROJECT ID DECIMAL(8) NOT NULL,"
```

```
& + "CUSTOMER ID VARCHAR(10) NOT NULL,"
```

- & + "CONTRACT ID VARCHAR(10) NOT NULL,"
- & + "CONTRACT_PROJECT_ID DECIMAL(3) NOT NULL,"
- & + "DESCRIPTION VARCHAR(60) NOT NULL,"
- & + "START DATE DATETIME NOT NULL,"
- & + "END_DATE DATETIME,"
- & + "CURRENT_STATUS DECIMAL(2) NOT NULL,"
- & + "STATUS DATE DATETIME NOT NULL,"
- & + "LEAD_CONSULTANT VARCHAR(15) NOT NULL,"
- & + "COORDINATOR VARCHAR(15) NOT NULL,"
- & + "APPROVED_BY_FIRST DECIMAL(3) NOT NULL,"
- & + "APPROVED BY LAST DECIMAL(3) NOT NULL,"
- & + "APPROVED DATE DATETIME NOT NULL,"
- & + "APPROVAL_LOCATION DECIMAL(2) NOT NULL,"
- & + "WORK_LOCATION_ID DECIMAL(2),"
- & + "CREATED_BY VARCHAR(15) NOT NULL,"
- & + "CREATED DATE DATETIME NOT NULL,"
- & + "MODIFIED_BY VARCHAR(15),"
- & + "MODIFIED_DATE DATETIME,"
- & + "TEXT VARCHAR(700),"
- & + "REPLICATION_KEY VARCHAR(20)"
- & + ")"

Field Loop Example 2 (Multi-Line and In-Line)

The following template file code:

```
sql = "INSERT INTO <STRUCTURE_NAME> ("
<FIELD_LOOP>
& + "<FIELD_SQLNAME><,>"
</FIELD_LOOP>
& + ") VALUES(<FIELD_LOOP>:<FIELD#LOGICAL><,>
</FIELD_LOOP>)"
```

Could produce output like this:

```
sql = "INSERT INTO PROJECT ("
& + "SYNERGYGRFA,"
& + "PROJECT ID,"
```

```
& + "CUSTOMER ID,"
& + "CONTRACT_ID,"
& + "CONTRACT PROJECT ID,"
& + "DESCRIPTION,"
& + "START DATE."
& + "END_DATE,"
& + "CURRENT_STATUS,"
& + "STATUS DATE,"
& + "LEAD CONSULTANT,"
& + "COORDINATOR,"
& + "APPROVED BY FIRST,"
& + "APPROVED_BY LAST,"
& + "APPROVED DATE,"
& + "APPROVAL LOCATION,"
& + "WORK LOCATION ID."
& + "CREATED_BY,"
& + "CREATED DATE,"
& + "MODIFIED_BY,"
& + "MODIFIED_DATE,"
& + "TEXT,"
& + "REPLICATION KEY"
& + ")
VALUES(:1,:2,:3,:4,:5,:6,:7,:8,:9,:10,:11,:12,:13,:14,:15,:16,:17,:18,:19,:20
```

Field Loop Example 3

The following template code:

```
#region Private fields (storage for properties)
<FIELD_LOOP>
private <FIELD_CSTYPE> p_<Field_Sqlname>;
</FIELD_LOOP>
#endregion
```

Could produce code like this:

```
#region Private fields (storage for properties)
private string p_Synergygrfa;
private int p_Project_Id;
```

```
private string p_Customer_Id;
private string p_Contract_Id;
private int p_Contract_Project_Id;
private string p_Description;
private DateTime p_Start_Date;
private DateTime p_End_Date;
private int p_Current_Status;
private DateTime p_Status_Date;
private string p_Lead_Consultant;
private string p_Coordinator;
private int p_Approved_By_First;
private int p_Approved_By_Last;
private DateTime p_Approved_Date;
private int p_Approval_Location;
private int p_Work_Location_Id;
private string p_Created_By;
private DateTime p_Created_Date;
private string p_Modified_By;
private DateTime p_Modified_Date;
private string p_Text;
private string p_Replication_Key;
#endregion
```

Video

Here's the Introduction to Field Loops video. Make sure you select a high-definition version of the video, by default YouTube tends to play the lowest resolution which can look pretty awful!

Field Loop Expansion Tokens

Field loop expansion tokens are used to insert information about the current field being processed within the context of a field loop into the output stream. Field loop tokens can only be used inside a field loop.

Field Loop Token	Description
<field#></field#>	Inserts the 1-based index of the current field in the record, without considering array elements or group fields as individual fields.
<field#_zero></field#_zero>	Inserts the 0-based index of the current field in the record, without considering array elements or group fields as individual fields.
<field#logical></field#logical>	Inserts the 1-based index of the current field in the record. Array elements and group fields are considered as individual fields. For example a structure with three fields, one of which is an array of 8 elements, is treated as having 10 fields.
<field#logical_zero></field#logical_zero>	Inserts the 0-based index of the current field in the record. Array elements and group fields are considered as individual fields. For example a structure with three fields, one of which is an array of 8 elements, is treated as having 10 fields.
<field_altname></field_altname>	Inserts the alternate name of the field. Array fields are represented in Synergy format, for example ALTNAME[1]. If no alternate name is available then the real name is used.
	Inserts the name of the fields arrive

<field_arrivem></field_arrivem>	method if present, or by nothing if not.
<field_basename></field_basename>	Inserts the base name of the field. Array field dimensions are not included in the base name.
<field_break_mode></field_break_mode>	Inserts a value indicating whether the field is a break field, and if so what type of break field.
<field_changem></field_changem>	Inserts the name of the fields change method if present, or by nothing if not.
<field_col></field_col>	Inserts the screen column for placement of the field on the screen in a cell-based environment. If the field has a column position specified then this is used, otherwise the default of column 15 is used.
<field_csconvert></field_csconvert>	Inserts a string indicating the appropriate C# string to numeric conversion method to use to attempt to parse a string into a numeric value.
<field_csdefault></field_csdefault>	Inserts a C# default value based on the fields data type.
<field_cstype></field_cstype>	Inserts a string indicating the C# equivalent data type of the field.
<field_default></field_default>	Inserts the fields default value, if present.
<field_desc></field_desc>	Inserts the description of the field.
	When CodeGen encounters array fields it "flattens" those arrays out into multiple fields each of which is of the same type and size. When processing a field which

<FIELD_DIMENSION1_INDEX> was formally part of an array, this token can be used to insert the original dimension two index of the field. For nonarray fields inserts a single zero (0).

<FIELD_DIMENSION2 INDEX</pre>

When CodeGen encounters array fields it "flattens" those arrays out into multiple fields each of which is of the same type and size. When processing a field which was formally part of an array, this token can be used to insert the original dimension three index of the field. For non-array fields inserts a single zero (0).

<FIELD_DIMENSION3 INDEX

When CodeGen encounters array fields it "flattens" those arrays out into multiple fields each of which is of the same type and size. When processing a field which was formally part of an array, this token can be used to insert the original dimension four index of the field. For non-array fields inserts a single zero (0).

<FIELD DIMENSION4 INDEX

When CodeGen encounters array fields it "flattens" those arrays out into multiple fields each of which is of the same type and size. When processing a field which was formally part of an array, this token can be used to insert the original dimension one index of the field. For nonarray fields inserts a single zero (0).

<FIELD DRILLM>

Inserts the name of the fields drill method if present, or by nothing if not.

<FIELD DRILL PIXEL COL>

Inserts the pixel column for placement of a fields drill button on a form.

<field_element></field_element>	For array fields inserts the one-based element number of the field currently being processed by the field loop. For multi-dimension arrays the token inserts a comma-separated list of current index numbers. For non-array fields the token does not cause anything to be written to the output stream.
<field_element0></field_element0>	For array fields inserts the one-based element number of the field currently being processed by the field loop. For multi-dimension arrays the token inserts a comma-separated list of current index numbers. For non-array fields the token inserts 0.
<field_enumlength></field_enumlength>	Inserts the maximum length of the fields' enumerated values.
<field_enumwidth></field_enumwidth>	Inserts the pixel width for the maximum length of the fields' enumerated values.
<field_formatname></field_formatname>	Inserts the name of the fields Repository format, or nothing if no format is specified.
<field_formatstring></field_formatstring>	Inserts the format string value of the fields Repository format, or nothing if no format is specified.
<field_heading></field_heading>	Inserts the fields report heading. If no heading exists then the field prompt is used. If no prompt exists then the fields SQL name is used. Array fields will have the element number appended (e.g. Phone #1).

<FIELD_HELPID>

Inserts the fields help identifier if present.

<field_hyperm></field_hyperm>	Inserts the name of the fields hyperlink method if present, or by nothing if not.
<field_infoline></field_infoline>	Inserts the fields' information line text if present.
<field_input_length></field_input_length>	Inserts the maximum number of characters that can be typed into a UI input field representing the current field.
<field_ldesc></field_ldesc>	Inserts the fields' long description.
<field_leavem></field_leavem>	Inserts the name of the fields leave method if present, or by nothing if not.
<field_maxvalue></field_maxvalue>	Inserts the maximum value which should be associated with a numeric field. The token should only be used for numeric fields, so you should only use it within an <if numeric=""> field loop expression.</if>
<field_minvalue></field_minvalue>	Inserts the minimum value which should be associated with a numeric field. The token should only be used for numeric fields, so you should only use it within an <if numeric=""> field loop expression.</if>
<field_name></field_name>	Inserts the name of the field. Array fields are represented in Synergy real array format, and group fields are represented by a full path.
<field_netname></field_netname>	Inserts a .NET compatible name for the current field. Array fields are expanded to multiple individual fields, with the name being suffixed with the array element number, for example 'FIELDNAME1', 'FIELDNAME2' etc.

<field_net_altname></field_net_altname>	Inserts a .NET compatible version of the alternate name for the current field.
<field_noecho_char></field_noecho_char>	Inserts the character that should be used to represent each character typed into a field, in place of the actual character (think password fields).
<field_ocdefault></field_ocdefault>	Inserts an Objective-C default value based on the fields data type.
<field_octype></field_octype>	Inserts a string indicating the Objective-C equivalent data type of the field.
<field_odbcname></field_odbcname>	Inserts an ODBC-compatible name for the current field. Array fields are expanded to multiple individual fields, with the name being suffixed with an underscore and the array element number, for example 'FIELDNAME_1', 'FIELDNAME_2' etc.
<field_original_name></field_original_name>	Inserts the original name of the field as defined in the repository. A fields original name never changes, and does not include any array dimensions or paths resulting from the expansion of array or group fields.
<field_path></field_path>	Inserts the path of the field. Array fields are represented in Synergy format, for example structure.fieldname[1].
<field_path_conv></field_path_conv>	Inserts the path of the field. Array fields are represented in Synergy format, for example structure.fieldname[1]. If the fields user text or long description contains an @UNMAPF=function; expression then the conversion function will also be applied.

<field_pixel_col></field_pixel_col>	Inserts the pixel column for placement of the field within a UI form. If the field has a column position specified then this is used, otherwise the default of column 15 is used.
<field_pixel_row></field_pixel_row>	Inserts the pixel row for placement of the field within a UI form. If the field has a row position defined then this is used, otherwise the logical field number (which includes array elements) is used to calculate the default row.
<field_pixel_width></field_pixel_width>	Inserts the pixel width for default sizing of the field when represented as a TextBox.
<field_position></field_position>	Inserts the one-based byte position of the field in the record.
<field_position_zero></field_position_zero>	Inserts the zero-based byte position of the field in the record.
<field_precision></field_precision>	Inserts the precision of an implied-decimal field or an empty string. Only the precision is inserted, with no leading period.
<field_precision2></field_precision2>	Inserts the precision of an implied- decimal field including a leading period, or an empty string.
<field_prompt></field_prompt>	Inserts the fields prompt.
<field_range_max></field_range_max>	Inserts the numeric fields' maximum value.
<field_range_min></field_range_min>	Inserts the numeric fields' minimum value.

<field_regex></field_regex>	Inserts a regular expression that can be used to constrain and validate user input to a field based on the repository definition of the field. The regular expression inserted depends on the type of field.
<field_row></field_row>	Inserts the screen row for placement of the field on the screen in a cell based environment. If the field has a row position defined then this is used, otherwise the logical field number (which includes array elements) is used to calculate the default row.
<field_sample_data></field_sample_data>	Inserts sample date for the field.
<pre><field_selection_count> <field_selections></field_selections></field_selection_count></pre>	Inserts the number of selection list values that are associated with the field. Inserts a comma separated list of the
TIEED_GEEEGTTOT(G	fields' selection list values.
<field_selections1></field_selections1>	Inserts a quoted string containing a pipe () delimited list of the fields selection list values.
<field_sellength></field_sellength>	Inserts the maximum length of the fields' selection list values.
<field_selwnd></field_selwnd>	Inserts the name of the selection window associated with the current field. If the field does not have a selection window then the token inserts nothing.
<field_size></field_size>	Inserts the size of the field in bytes.
<field_sndefault></field_sndefault>	Inserts a Synergy .NET default value based on the fields data type.

<field_sntype></field_sntype>	Inserts a string indicating the Synergy NET data type of the field.
<field_spec></field_spec>	Inserts a Synergy data type definition for the field.
<field_sqlname></field_sqlname>	Inserts a SQL-compatible name for the current field. Array fields are expanded to multiple individual fields, with the name being suffixed with the array element number, for example 'FIELDNAME1', 'FIELDNAME2' etc.
<field_sql_altname></field_sql_altname>	Inserts a SQL-compatible version of the alternate name for the current field.
<field_sqltype></field_sqltype>	Inserts a SQL compatible data definition of the field.
<field_template></field_template>	Inserts the name of the Repository template from which the field inherited its attributes or by a null string if no template is referenced.
<field_tkscript></field_tkscript>	Inserts a UI Toolkit .field script file command for the field.
<field_type></field_type>	Inserts a character indicating the synergy data type of field.
<field_type_name></field_type_name>	Inserts a string indicating the Synergy data type of field.
<field_utext></field_utext>	Inserts the fields' user-defined text string.
<field_vbdefault></field_vbdefault>	Inserts a VB.NET default value based on the fields data type.
<field_vbtype></field_vbtype>	Inserts a string indicating the VB.NET

data type of the field.

<MAPPED_FIELD>

Inserts the name of the mapped field in the mapped structure. To use this token you must use @MAP=structure; in the structures user text field or long description and @MAP=field; in each fields user text field or long description.

<MAPPED_PATH>

Inserts the name of the mapped path in the mapped structure. To use this token you must use @MAP=structure; in the structures user text field or long description and @MAP=field; in each fields user text field or long description.

<MAPPED PATH CONV>

Inserts the name of the mapped path in the mapped structure. To use this token you must use @MAP=structure; in the structures user text field or long description and @MAP=field; in each fields user text field. If there is also an @MAPF=function; in the user text or long description then that conversion function will also be applied.

<PROMPT_COL>

Inserts the screen column for placement of the prompt on the screen in a character based environment. If the prompt has a column position specified then this is used, otherwise the default of column 1 is used.

<PROMPT_PIXEL_COL>

Inserts the pixel column for placement of the prompt on a windows form. If the prompt has a column position specified then this is used, otherwise the default of column 1 is used. <PROMPT_PIXEL_ROW>

Inserts the pixel row for placement of the prompt on a windows form. If the prompt has a row position defined then this is used, otherwise the logical field number (which includes array elements) is used to calculate the default row.

<PROMPT_PIXEL_WIDTH>

Inserts the pixel width for default sizing of the prompt when represented as a label.

<PROMPT_ROW>

Inserts the screen row for placement of the prompt on the screen in a character based environment. If the prompt has a row position defined then this is used, otherwise the logical field number (which includes array elements) is used to calculate the default row.

FIELD#

Inserts the 1-based index of the current field in the record, without considering array elements or group fields as individual fields.

Usage

<FIELD#>

Alternate forms

None

See also

- <FIELD#_ZERO>
- <FIELD#LOGICAL>
- <FIELD#LOGICAL_ZERO>
- Other field loop expansion tokens

FIELD#_ZERO

Inserts the 0-based index of the current field in the record, without considering array elements or group fields as individual fields.

Usage

<FIELD#_ZERO>

Alternate forms

None

See also

- <FIELD#>
- <FIELD#LOGICAL>
- <FIELD#LOGICAL_ZERO>
- Other field loop expansion tokens

FIELD#LOGICAL

Inserts the 1-based index of the current field in the record. Array elements and group fields are considered as individual fields. For example a structure with three fields, one of which is an array of 8 elements, is treated as having 10 fields.

Usage

<FIELD#LOGICAL>

Alternate forms

None

See also

- <FIELD#>
- <FIELD#_ZERO>
- <FIELD#LOGICAL_ZERO>
- Other field loop expansion tokens

FIELD#LOGICAL_ZERO

Inserts the 0-based index of the current field in the record. Array elements and group fields are considered as individual fields. For example a structure with three fields, one of which is an array of 8 elements, is treated as having 10 fields.

Usage

<FIELD#LOGICAL_ZERO>

Alternate forms

None

See also

- <FIELD#>
- <FIELD#_ZERO>
- <FIELD#LOGICAL>
- Other field loop expansion tokens

FIELD_ALTNAME

Inserts the alternate name of the field. Array fields are represented in Synergy format, for example ALTNAME[1]. If no alternate name is available then the real name is used.

Usage

<FIELD_ALTNAME>

Alternate forms

- <field_altname>
- <Field altname>
- <Field_Altname>
- <FieldAltname>
- <fieldAltname>

See also

- <FIELD_BASENAME>
- <FIELD_NAME>
- <FIELD_NETNAME>
- <FIELD_NET_ALTNAME>
- <FIELD_ODBCNAME>
- <FIELD_ORIGINAL_NAME>
- <FIELD_SQL_ALTNAME>
- <FIELD_SQLNAME>
- Other field loop expansion tokens

FIELD_ARRIVEM

Inserts the name of the fields arrive method if present, or by nothing if not.

Usage

<FIELD_ARRIVEM>

Alternate forms

- <field_arrivem>
- <Field_Arrivem>
- <Field_arrivem>
- <FieldArrivem>
- <fieldArrivem>

See also

- <FIELD_CHANGEM>
- <FIELD_DRILLM>
- <FIELD_HYPERM>
- <FIELD_LEAVEM>
- Other field loop expansion tokens

FIELD_BASENAME

Inserts the base name of the field. Array field dimensions are not included in the base name.

Usage

<FIELD_BASENAME>

Alternate forms

- <field basename>
- <Field_Basename>
- <Field_basename>
- <FieldBaseName>
- <fieldBaseName>

See also

- <FIELD_ALTNAME>
- <FIELD_NAME>
- <FIELD_NETNAME>
- <FIELD_NET_ALTNAME>
- <FIELD_ODBCNAME>
- <FIELD_ORIGINAL_NAME>
- <FIELD_SQL_ALTNAME>
- <FIELD_SQLNAME>
- Other field loop expansion tokens

FIELD_BREAK_MODE

Inserts a value indicating whether the field is a break field, and if so what type of break field.

Usage

<FIELD_BREAK_MODE>

Possible Values

- None
- Change
- Always
- Return

Alternate forms

none

See also

• Other field loop expansion tokens

FIELD_CHANGEM

Inserts the name of the fields change method if present, or by nothing if not.

Usage

<FIELD_CHANGEM>

Alternate forms

- <field_changem>
- <Field_Changem>
- <Field_changem>
- <FieldChangem>
- <fieldChangem>

See also

- <FIELD_ARRIVEM>
- <FIELD_DRILLM>
- <FIELD_HYPERM>
- <FIELD_LEAVEM>
- Other field loop expansion tokens

FIELD COL

Inserts the screen column for placement of the field on the screen in a cell-based environment. If the field has a column position specified then this is used, otherwise the default of column 15 is used.

Usage

<FIELD COL>

Alternate forms

None

See also

- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_PIXEL_WIDTH>
- <FIELD ROW>
- <PROMPT_COL>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_PIXEL_WIDTH>
- <PROMPT_ROW>
- Other field loop expansion tokens

FIELD_CSCONVERT

Inserts a string indicating the appropriate C# string to numeric conversion method to use to attempt to parse a string into a numeric value.

For non-numeric fields the token is replaced by nothing.

Usage

<FIELD_CSCONVERT>

Possible values

- bool.TryParse
- byte.TryParse
- decimal.TryParse
- double.TryParse
- int.TryParse
- float.TryParse
- long.TryParse
- sbyte.TryParse
- short.TryParse
- uint.TryParse
- ulong.TryParse
- ushort.TryParse

Alternate forms

None

Example of use

if (!<FIELD_CSCONVERT>

(recordInString.Substring(<FIELD_POSITION_ZERO>,<FIELD_SIZE>),out
p_<Field_Sqlname>))

See also

- <FIELD_CSDEFAULT>
- <FIELD_CSTYPE>
- Other field loop expansion tokens

FIELD_CSDEFAULT

Inserts a C# default value based on the fields data type.

Usage

<FIELD_CSDEFAULT>

Possible values

- ""
- 0
- false
- new DateTime()
- null

Alternate forms

None

See also

- <FIELD_CSCONVERT>
- <FILED_CSTYPE>
- Other field loop expansion tokens

FIELD_CSTYPE

Inserts a string indicating the C# equivalent data type of the field.

Usage

<FIELD_CSTYPE>

Possible values

- bool
- byte
- double
- float
- int
- long
- decimal
- DateTime
- Nullable<DateTime>
- sbyte
- short
- string
- uint
- ulong
- ushort

Alternate forms

None

See also

- <FIELD_CSCONVERT>
- <FIELD_CSDEFAULT>
- Other field loop expansion tokens

FIELD_DEFAULT

Inserts the fields default value, if present.

Usage

<FIELD_DEFAULT>

Alternate forms

None

See also

• Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_DESC

Inserts the description of the field.

Usage

<FIELD_DESC>

Alternate forms

None

See also

- <FIELD_INFOLINE>
- <FIELD_LDESC>
- Other field loop expansion tokens

FIELD_DIMENSION1_INDEX

When CodeGen encounters array fields it "flattens" those arrays out into multiple fields each of which is of the same type and size. When processing a field which was formally part of an array, this token can be used to insert the original dimension one index of the field. For non-array fields inserts a single zero (0).

Usage

<FIELD_DIMENSION1_INDEX>

Alternate forms

None

See also

- <FIELD_DIMENSION2_INDEX>
- <FIELD_DIMENSION3_INDEX>
- <FIELD_DIMENSION4_INDEX>
- <FIELD_ELEMENT>
- <FIELD_ELEMENT0>
- Other field loop expansion tokens

FIELD_DIMENSION2_INDEX

When CodeGen encounters array fields it "flattens" those arrays out into multiple fields each of which is of the same type and size. When processing a field which was formally part of an array with at least two dimensions, this token can be used to insert the original dimension two index of the field. For non-array fields or for array fields with only one dimension inserts a single zero (0).

Usage

<FIELD_DIMENSION2_INDEX>

Alternate forms

None

See also

- <FIELD_DIMENSION1_INDEX>
- <FIELD_DIMENSION3_INDEX>
- <FIELD_DIMENSION4_INDEX>
- <FIELD_ELEMENT>
- <FIELD_ELEMENT0>
- Other field loop expansion tokens

FIELD_DIMENSION3_INDEX

When CodeGen encounters array fields it "flattens" those arrays out into multiple fields each of which is of the same type and size. When processing a field which was formally part of an array with at least three dimensions, this token can be used to insert the original dimension three index of the field. For non-array fields or for array fields with less than three dimensions inserts a single zero (0).

Usage

<FIELD_DIMENSION3_INDEX>

Alternate forms

None

See also

- <FIELD_DIMENSION1_INDEX>
- <FIELD_DIMENSION2_INDEX>
- <FIELD_DIMENSION4_INDEX>
- <FIELD_ELEMENT>
- <FIELD_ELEMENT0>
- Other field loop expansion tokens

FIELD_DIMENSION4_INDEX

When CodeGen encounters array fields it "flattens" those arrays out into multiple fields each of which is of the same type and size. When processing a field which was formally part of an array with four dimensions, this token can be used to insert the original dimension four index of the field. For non-array fields or for array fields with less than four dimensions inserts a single zero (0).

Usage

<FIELD_DIMENSION4_INDEX>

Alternate forms

None

See also

- <FIELD_DIMENSION1_INDEX>
- <FIELD_DIMENSION2_INDEX>
- <FIELD_DIMENSION3_INDEX>
- <FIELD_ELEMENT>
- <FIELD_ELEMENT0>
- Other field loop expansion tokens

FIELD_DRILL_PIXEL_COL

Inserts the pixel column for placement of a fields drill button on a form.

Usage

<FIELD_DRILL_PIXEL_COL>

Example Output

275

Alternate forms

None

- <FIELD_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_ROW>
- <PROMPT_COL>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_PIXEL_WIDTH>
- <PROMPT_ROW>
- Other field loop expansion tokens

FIELD_DRILLM

Inserts the name of the fields drill method if present, or by nothing if not.

Usage

<FIELD_DRILLM>

Alternate forms

- <field_drillm>
- <Field_Drillm>
- <Field_drillm>
- <FieldDrillm>
- <fieldDrillm>

See also

- <FIELD_ARRIVEM>
- <FIELD_CHANGEM>
- <FIELD_HYPERM>
- <FIELD_LEAVEM>
- Other field loop expansion tokens

Copyright @ 2018 Synergex International, Inc.

FIELD ELEMENT

For array fields inserts the one-based element number of the field currently being processed by the field loop. For multi-dimension arrays the token inserts a comma-separated list of current index numbers. For non-array fields the token does not cause anything to be written to the output stream.

Usage

<FIELD_ELEMENT>

Example Output

For a non-array field: <no output>

For a single-dimensional array: 1

For a two-dimensional array: 1,2

For a three-dimensional array: 1,2,3

For a four-dimensional array: 1,2,3,4

Alternate forms

None

- <FIELD_DIMENSION1_INDEX>
- <FIELD_DIMENSION2_INDEX>
- <FIELD_DIMENSION3_INDEX>
- <FIELD_DIMENSION4_INDEX>
- <FIELD_ELEMENT0>
- Other field loop expansion tokens

FIELD ELEMENTO

For array fields inserts the one-based element number of the field currently being processed by the field loop. For multi-dimension arrays the token inserts a comma-separated list of current index numbers. For non-array fields the token inserts 0.

Usage

<FIELD_ELEMENT0>

Example Output

For a non-array field: 0

For a single-dimensional array: 1

For a two-dimensional array: 1,2

For a three-dimensional array: 1,2,3

For a four-dimensional array: 1,2,3,4

Alternate forms

None

- <FIELD_DIMENSION1_INDEX>
- <FIELD_DIMENSION2_INDEX>
- <FIELD_DIMENSION3_INDEX>
- <FIELD_DIMENSION4_INDEX>
- <FIELD ELEMENT>
- Other field loop expansion tokens

FIELD_ENUMLENGTH

Inserts the maximum length of the fields' enumerated values.

Usage

<FIELD_ENUMLENGTH>

Example Output

12

Alternate forms

None

See also

- <FIELD_ENUMWIDTH>
- Other field loop expansion tokens

FIELD_ENUMWIDTH

Inserts the pixel width for the maximum length of the fields' enumerated values.

Calculated as the maximum length of the fields enumerated values multiplied by 12 pixels per character. This default character width can be overridden using the -cw command line option.

Usage

<FIELD_ENUMWIDTH>

Example Output

144

Alternate forms

None

See also

- <FIELD_ENUMLENGTH>
- Other field loop expansion tokens

FIELD_FORMATNAME

Inserts the name of the fields Repository format, or nothing if no format is specified.

Usage

<FIELD_FORMATNAME>

Alternate forms

None

See also

- <FIELD_FORMATSTRING>
- Other field loop expansion tokens

FIELD_FORMATSTRING

Inserts the format string value of the fields Repository format, or nothing if no format is specified.

Usage

<FIELD_FORMATSTRING>

Alternate forms

None

See also

- <FIELD_FORMATNAME>
- Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_HEADING

Inserts the fields report heading. If no heading exists then the field prompt is used. If no prompt exists then the fields SQL name is used. Array fields will have the element number appended (e.g. Phone #1).

Usage

<FIELD_HEADING>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD_HELPID

Inserts the fields help identifier if present.

Usage

<FIELD_HELPID>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD_HYPERM

Inserts the name of the fields' hyperlink method if present, or by nothing if not.

Usage

<FIELD_HYPERM>

Alternate forms

- <field_hyperm>
- <Field_Hyperm>
- <Field_hyperm>
- <FieldHyperm>
- <fieldHyperm>

See also

- <FIELD_ARRIVEM>
- <FIELD_CHANGEM>
- <FIELD_DRILLM>
- <FIELD_LEAVEM>
- Other field loop expansion tokens

FIELD_INFOLINE

Inserts the fields' information line text if present.

Usage

<FIELD_INFOLINE>

Alternate forms

None

See also

- <FIELD_DESC>
- <FIELD_LDESC>
- Other field loop expansion tokens

FIELD INPUT LENGTH

Inserts the maximum number of characters that can be typed into a UI input field representing the current field.

- For alpha fields the input length is the same as the field size.
- For decimal fields the input length will be one byte longer than the field size if the field allows negative values.
- For implied decimal fields the input length will be one byte longer than the field size to allow for the entry of a decimal point, and two bytes longer that the field size if the field allows negative values.
- For integer fields the input length is based on the maximum number of bytes that are required in order to be able to enter the maximum value supported by the integer field, but bear in mind that this may allow your user to enter a value larger than that supported by the underlying field. For example, an I1 field which allows negative values is capable of storing values in the range of -128 to 127, so the input length for the field is 4, but when used to constrain the length of a UI field this would allows the user to enter a value in the range -999 to 9999.

Usage

<FIELD_INPUT_LENGTH>

Alternate forms

None

- <FIELD_SIZE>
- Other field loop expansion tokens

FIELD_LDESC

Inserts the fields' long description.

Usage

<FIELD_LDESC>

Alternate forms

None

See also

- <FIELD_DESC>
- <FIELD_INFOLINE>
- Other field loop expansion tokens

FIELD_LEAVEM

Inserts the name of the fields leave method if present, or by nothing if not.

Usage

<FIELD_LEAVEM>

Alternate forms

- <field_leavem>
- <Field_Leavem>
- <Field_leavem>
- <FieldLeavem>
- <fieldLeavem>

See also

- <FIELD_ARRIVEM>
- <FIELD_CHANGEM>
- <FIELD_DRILLM>
- <FIELD_HYPERM>
- Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_MAXVALUE

Inserts the maximum value which should be associated with a numeric field. The token should only be used for numeric fields, so you should only use it within an **<IF NUMERIC>** field loop expression.

Usage

<FIELD_MAXVALUE>

Alternate forms

None

See also

- <FIELD_MINVALUE>
- Other field loop expansion tokens

FIELD_MINVALUE

Inserts the minimum value which should be associated with a numeric field. The token should only be used for numeric fields, so you should only use it within an **IF NUMERIC** field loop expression.

Usage

<FIELD_MINVALUE>

Alternate forms

None

See also

- <FIELD_MAXVALUE>
- Other field loop expansion tokens

FIELD_NAME

Inserts the name of the field. Array fields are represented in Synergy real array format, and group fields are represented by a full path.

Usage

<FIELD_NAME>

Example Output

- ACCOUNT_NUMBER
- PHONE[3]
- ADDRESS.CITY
- CONTACT[2].NAME

Alternate forms

- <field_name>
- <Field_Name>
- <Field name>
- <FieldName>
- <fieldName>

- <FIELD_ALTNAME>
- <FIELD_BASENAME>
- <FIELD_NETNAME>
- <FIELD_NET_ALTNAME>
- <FIELD_ODBCNAME>

- <FIELD_ORIGINAL_NAME>
- <FIELD_SQL_ALTNAME>
- <FIELD_SQLNAME>
- Other field loop expansion tokens

FIELD NETNAME

Inserts a .NET compatible name for the current field. Array fields are expanded to multiple individual fields, with the name being suffixed with the array element number, for example 'FIELDNAME1', 'FIELDNAME2' etc.

The token behaves identically to the <FIELD_SQLNAME> token.

Usage

<FIELD_NETNAME>

Example Output

- ACCOUNT_NUMBER
- PHONE3
- ADDRESS CITY
- CONTACT2_NAME

Alternate forms

- <field netname>
- <Field_Netname>
- <Field_netname>
- <FieldNetname>
- <fieldNetname>

In addition, for compatibility with previous earlier versions, the following alternate forms are also supported but ideally should not be used:

- <FieldNetName>
- <fieldNetName>

- <FIELD_ALTNAME>
- <FIELD_BASENAME>
- <FIELD_NAME>
- <FIELD_NET_ALTNAME>
- <FIELD_ODBCNAME>
- <FIELD_ORIGINAL_NAME>
- <FIELD_SQL_ALTNAME>
- <FIELD_SQLNAME>
- Other field loop expansion tokens

FIELD_NET_ALTNAME

Inserts a .NET compatible version of the alternate name for the current field. Array fields are expanded to multiple individual fields, with the name being suffixed with the array element number, for example 'FIELDNAME1', 'FIELDNAME2' etc.

The token behaves identically to the <FIELD_SQL_ALTNAME> token.

Usage

<FIELD_NET_ALTNAME>

Alternate forms

- <field net altname>
- <Field_Net_Altname>
- <Field_net_altname>
- <FieldNetAltname>
- <fieldNetAltname>

- <FIELD_ALTNAME>
- <FIELD_BASENAME>
- <FIELD_NAME>
- <FIELD_NETNAME>
- <FIELD_ODBCNAME>
- <FIELD_ORIGINAL_NAME>

- <FIELD_SQL_ALTNAME>
- <FIELD_SQLNAME>
- Other field loop expansion tokens

FIELD_NOECHO_CHAR

Inserts the character that should be used to represent each character typed into a field, in place of the actual character (think password fields).

If the current field being processed is a "no echo" field then the token will be replaced by the no echo character specified in the repository field definition, or by an asterisk (*) if no character is specified. If the field is not a "no echo" field then the token will be replaced by a null string.

Usage

<FIELD_NOECHO_CHAR>

Alternate forms

None

See also

- <IF ECHO>
- <IF NOECHO>
- Other field loop expansion tokens

FIELD_OCDEFAULT

Inserts an Objective-C default value based on the fields data type.

Usage

<FIELD_OCDEFAULT>

Possible values

- @""
- 0
- 0.0
- NO
- [NSDate new]

Alternate forms

None

See also

- <FIELD_OCTYPE>
- Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_OCTYPE

Inserts a string indicating the Objective-C equivalent data type of the field.

Usage

<FIELD_OCTYPE>

Possible values

- BOOL
- double
- float
- int
- long int
- NSDate
- NSString
- short int
- unsigned int
- unsigned long int
- unsigned short int

Alternate forms

None

- <FIELD_OCDEFAULT>
- Other field loop expansion tokens

FIELD_ODBCNAME

Inserts an ODBC-compatible name for the current field. Array fields are expanded to multiple individual fields, with the name being suffixed with an underscore and the array element number, for example 'FIELDNAME_1', 'FIELDNAME_2' etc. Note however that using the alternate pascal-case (<FieldOdbcname>) or camel-case (<fieldOdbcname>) tokens will cause the underscore characters to be suppressed.

Usage

<FIELD_ODBCNAME>

Alternate forms

- <field_odbcname>
- <Field_Odbcname>
- <Field odbcname>
- <FieldOdbcname>
- <fieldOdbcname>

In addition, for compatibility with previous earlier versions, the following alternate forms are also supported but ideally should not be used:

- <FieldOdbcName>
- <fieldOdbcName>

- <FIELD ALTNAME>
- <FIELD BASENAME>
- <FIELD NAME>
- <FIELD_NETNAME>

- <FIELD_NET_ALTNAME>
- <FIELD_ORIGINAL_NAME>
- <FIELD_SQL_ALTNAME>
- <FIELD_SQLNAME>
- Other field loop expansion tokens

FIELD_ORIGINAL_NAME

Inserts the original name of the field as defined in the repository. A fields original name never changes, and does not include any array dimensions or paths resulting from the expansion of array or group fields.

Usage

<FIELD_ORIGINAL_NAME>

Alternate forms

- <field_original_name>
- <Field_Original_Name>
- <Field_original_name>
- <FieldOriginalName>
- <fieldOriginalName>

- <FIELD_ALTNAME>
- <FIELD_BASENAME>
- <FIELD_NAME>
- <FIELD_NETNAME>
- <FIELD_NET_ALTNAME>
- <FIELD_ODBCNAME>
- <FIELD_SQL_ALTNAME>
- <FIELD_SQLNAME>
- Other field loop expansion tokens

FIELD_PATH

Inserts the path of the field. Array fields are represented in Synergy format, for example structure.fieldname[1].

Usage

<FIELD_PATH>

Alternate forms

- <field_path>
- <Field_path>
- <Field_Path>
- <FieldPath>
- <fieldPath>

See also

• Other field loop expansion tokens

FIELD_PATH_CONV

Inserts the path of the field. Array fields are represented in Synergy format, for example structure.fieldname[1]. If the fields user text or long description contains an @UNMAPF=function; expression then the conversion function will also be applied.

Usage

<FIELD_PATH_CONV>

Alternate forms

<field_path_conv>

See also

• Other field loop expansion tokens

FIELD_PIXEL_COL

Inserts the pixel column for placement of the field within a UI form. If the field has a column position specified then this is used, otherwise the default of column 15 is used.

A default width of 12 pixels per character is used, and this can be overridden using the -cw command line option.

Usage

<FIELD_PIXEL_COL>

Alternate forms

None

- <FIELD_COL>
- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_PIXEL_WIDTH>
- <FIELD_ROW>
- <PROMPT_COL>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_PIXEL_WIDTH>
- <PROMPT_ROW>



FIELD PIXEL ROW

Inserts the pixel row for placement of the field within a UI form. If the field has a row position defined then this is used, otherwise the logical field number (which includes array elements) is used to calculate the default row.

A default height of 25 pixels per row is used, and this can be overridden using the -ch command line option.

Usage

<FIELD PIXEL ROW>

Alternate forms

None

- <FIELD_COL>
- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_WIDTH>
- <FIELD ROW>
- <PROMPT_COL>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_PIXEL_WIDTH>
- <PROMPT_ROW>



FIELD_PIXEL_WIDTH

Inserts the pixel width for default sizing of the field when represented as a TextBox.

Calculated as the size of the field in characters multiplied by 12 pixels per character. This default character width can be overridden using the -cw command line option.

Usage

<FIELD_PIXEL_WIDTH>

Alternate forms

None

- <FIELD_COL>
- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_ROW>
- <PROMPT_COL>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_PIXEL_WIDTH>
- <PROMPT_ROW>



FIELD_POSITION

Inserts the one-based byte position of the field in the record.

Usage

<FIELD_POSITION>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD_POSITION_ZERO

Inserts the zero-based byte position of the field in the record.

Usage

<FIELD_POSITION_ZERO>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD_PRECISION

Inserts the precision of an implied-decimal field or an empty string. Only the precision is inserted, with no leading period.

Usage

<FIELD_PRECISION>

Alternate forms

None

See also

- <FIELD_PRECISION0>
- <FIELD_PRECISION2>
- Other field loop expansion tokens

FIELD_PRECISION0

Inserts the precision of an implied-decimal field or 0 for non implied decimal fields.

Usage

<FIELD_PRECISION0>

Alternate forms

None

See also

- <FIELD_PRECISION>
- <FIELD_PRECISION2>
- Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_PRECISION2

Inserts the precision of an implied-decimal field including a leading period, or an empty string.

Usage

<FIELD_PRECISION2>

Alternate forms

None

See also

- <FIELD_PRECISION>
- <FIELD_PRECISION0>
- Other field loop expansion tokens

FIELD_PROMPT

Inserts the fields prompt.

Usage

<FIELD_PROMPT>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD_RANGE_MAX

Inserts the numeric fields' maximum value.

Usage

<FIELD_RANGE_MAX>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD_RANGE_MIN

Inserts the numeric fields' minimum value.

Usage

<FIELD_RANGE_MIN>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD REGEX

Inserts a regular expression that can be used to constrain and validate user input to a field based on the repository definition of the field. The regular expression inserted depends on the type of field.

Usage

<FIELD_REGEX>

Examples

Nature

of field Regular Description Expression

A4

^.{0,4}\$ Allows between 0 and 4 characters.

A4,

required

field ^.{1,4}\$ Allows between 1 and 4 characters.

D4, no Allows an optional + sign, followed by any number of negatives $^{[+]}?[0]*$ leading zeros, followed by up to four numeric digits. $[0-9]\{0,4\}$ \$

D4, Allows an optional + or - sign, followed by any number of leading zeros, followed by up to four numeric digits. $[0-9]{0,4}$ \$

Allows an optional + sign, followed by any number of leading zeros, followed by up to two numeric digits,

negatives $[0-9]\{0,2\}$ followed by an optional decimal point. If a decimal (\.[0-9] point is found then up to two numeric digits may follow, $\{0,2\}[0]^*$)? followed by any number of trailing zeros.

Allows an optional + or - sign, followed by any number D4.2, $^{[+-]?[0]*}$ of leading zeros, followed by up to two numeric digits, allows $[0-9]\{0,2\}$ followed by an optional decimal point. If a decimal negatives (\.[0-9] point is found then up to two numeric digits may follow, $\{0,2\}[0]^*$)? followed by any number of trailing zeros.

I4, no
$$\uparrow$$
[+]?[0]* Allows an optional + sign, followed by any number of leading zeros, followed by up to ten numeric digits. {0,10}\$

I4, allows
$$^{[+-]?[0]*}$$
 Allows an optional + or - sign, followed by any number of leading zeros, followed by up to ten numeric digits. $\{0,10\}$ \$

In the table above, the numbers shown in red are based on the size of the field being processed at the time.

Restrictions

Integer Fields

The regular expression for integer fields will constrain the number of characters entered, and restrict those characters to being numeric characters, but it will NOT restrict the user to only entering valid numeric values. For example, an i1 field can store up to 127, the regular expression will constrain the entry to 3 numeric characters but it will still be possible to enter an out of range value (e.g. 256).

Required Fields

Currently CodeGen only takes a fields "required" status into account for alpha fields. If an alpha field is required then the regular expression produced will require at least one character be present.

Alternate forms

None

See also

Other field loop expansion tokens

FIELD ROW

Inserts the screen row for placement of the field on the screen in a cell based environment. If the field has a row position defined then this is used, otherwise the logical field number (which includes array elements) is used to calculate the default row.

Usage

<FIELD_ROW>

Alternate forms

None

- <FIELD_COL>
- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_PIXEL_WIDTH>
- <PROMPT_COL>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_PIXEL_WIDTH>
- <PROMPT_ROW>
- Other field loop expansion tokens

FIELD_SAMPLE_DATA

Inserts sample date for the field. This token is intended to be used for the generation of examples in documentation or API references.

NOTE: This token only currently works with basic field types alpha, decimal, implied decimal, integer and boolean. For other field types nothing is inserted.

Usage

<FIELD SAMPLE DATA>

Possible Values

"ABC"

123

1.23

True

"Sample"

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD_SELECTION_COUNT

Inserts the number of selection list values that are associated with the field, for example:

4

A value of 0 will be inserted for fields that do not have associated selection lists.

Usage

<FIELD_SELECTION_COUNT>

Alternate forms

None

See also

- <FIELD_SELECTIONS>
- <FIELD_SELECTIONS1>
- Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_SELECTIONS

Inserts a comma separated list of the fields' selection list values, for example:

"Red","Green","Blue"

Usage

<FIELD_SELECTIONS>

Alternate forms

None

See also

- <FIELD_SELECTION_COUNT>
- <FIELD_SELECTIONS1>
- Other field loop expansion tokens

FIELD_SELECTIONS1

Inserts a quoted string containing a pipe (|) delimited list of the fields selection list values, for example:

"Red|Green|Blue"

Usage

<FIELD_SELECTIONS1>

Alternate forms

None

See also

- <FIELD_SELECTION_COUNT>
- <FIELD_SELECTIONS>
- Other field loop expansion tokens

FIELD_SELLENGTH

Inserts the maximum length of the fields' selection list values.

Usage

<FIELD_SELLENGTH>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD SELWND

Inserts the name of the selection window associated with the current field. If the field does not have a selection window then the token inserts nothing.

Bear in mind that if the -ws command line option is being used then CodeGen may have removed the selection window name from the field definition and replaced it with a selection list, based on the selection windows entries. In that case you can use a <SELECTION_LOOP> to process the selections.

Usage

<FIELD_SELWND>

Alternate forms

- <field_selwnd>
- <Field_Selwnd>
- <Field_selwnd>
- <FieldSelWnd>
- <FieldSelWnd>
- <FIELD_SELWND_ORIGINAL>

Note that in repository the name of a fields selection window can be entered in a mixed case format. The <FIELD_SELWND_ORIGINAL> token allows you to use the name of the selection window exactly as it appears in the repository.

- <IF SELWND>
- <IF NOSELWND>
- Other field loop expansion tokens



FIELD_SIZE

Inserts the size of the field in bytes.

Usage

<FIELD_SIZE>

Alternate forms

None

See also

- <FIELD_INPUT_LENGTH>
- Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_SNDEFAULT

Inserts a Synergy .NET default value based on the fields data type.

Usage

<FIELD_SNDEFAULT>

Possible values

- ""
- 0
- false
- new DateTime()
- ^null

Alternate forms

None

See also

- <FIELD_SNTYPE>
- Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_SNTYPE

Inserts a string indicating the Synergy NET data type of the field.

Usage

<FIELD_SNTYPE>

Possible values

- boolean
- byte
- DateTime
- decimal
- double
- float
- int
- long
- @Nullable<DateTime>
- short
- String
- ystem.SByte
- System.UInt16
- System.UInt32

• System.UInt64

Alternate forms

None

See also

- <FIELD_SNDEFAULT>
- Other field loop expansion tokens

FIELD SPEC

Inserts a Synergy data type definition for the field. For example:

A15

D12.4

BOOLEAN

Use with Implicit Groups

An implicit group is a group in the repository structure being processed whose fields are defined by another repository structure.

By default CodeGen will expand group fields to multiple individual fields based on the fields within the group that was encountered. As an alternative, if the -g i command line option is used (don't expand implicit groups to individual fields), and an implicit group is encountered, then the implicit group is assumed to be represented as a class, and the field specification will be inserted as a class type. For example, for an implicit group based on the repository structure CUSTOMER, the field specification will be:

@CUSTOMER

Usage

<FIELD_SPEC>

Alternate forms

- <field_spec>
- <Field_Spec>
- <Field_spec>
- <FieldSpec>

<fieldSpec>

See also

- <FIELD_CSTYPE>
- <FIELD_OCTYPE>
- <FIELD_SQLTYPE>
- <FIELD_TYPE>
- <FIELD_TYPE_NAME>
- <FIELD_VBTYPE>
- Other field loop expansion tokens

FIELD_SQL_ALTNAME

Inserts a SQL-compatible version of the alternate name for the current field. Array fields are expanded to multiple individual fields, with the name being suffixed with the array element number, for example 'FIELDNAME1', 'FIELDNAME2' etc.

The token behaves identically to the <FIELD_NET_NETNAME> token.

Usage

<FIELD_SQL_ALTNAME>

Alternate forms

- <field_sql_altname>
- <Field_Sql_Altname>
- <Field_sql_altname>
- <FieldSqlAltname>
- <fieldSqlAltname>

- <FIELD_ALTNAME>
- <FIELD_BASENAME>
- <FIELD_NAME>
- <FIELD_NETNAME>
- <FIELD_NET_ALTNAME>
- <FIELD_ODBCNAME>

- <FIELD_ORIGINAL_NAME>
- <FIELD_SQLNAME>
- Other field loop expansion tokens

FIELD_SQLNAME

Inserts a SQL-compatible name for the current field. Array fields are expanded to multiple individual fields, with the name being suffixed with the array element number, for example 'FIELDNAME1', 'FIELDNAME2' etc.

The token behaves identically to the <FIELD_NETNAME> token.

Usage

<FIELD_SQLNAME>

Alternate forms

- <field_sqlname>
- <Field_Sqlname>
- <Field_sqlname>
- <FieldSqlname>
- <fieldSqlname>

In addition, for compatibility with previous earlier versions, the following alternate forms are also supported but ideally should not be used:

- <FieldSqlName>
- <fieldSqlName>

- <FIELD_ALTNAME>
- <FIELD_BASENAME>

- <FIELD_NAME>
- <FIELD_NETNAME>
- <FIELD_NET_ALTNAME>
- <FIELD_ODBCNAME>
- <FIELD_ORIGINAL_NAME>
- <FIELD_SQL_ALTNAME>
- Other field loop expansion tokens

FIELD_SQLTYPE

Inserts a SQL compatible data definition of the field. By default the data types for Microsoft SQL Server are used but you can select one of the other supported databases either by using the -database command line option, or by setting the CODEGEN_DATABASE_TYPE environment variable.

Usage

<FIELD_SQLTYPE>

Alternate forms

None

Resulting Values

Synergy Data Type	SQL Server Type	MySQL Type	PostgreSQL Type
Alpha	VARCHAR(n)	VARCHAR(n)	VARCHAR(n)
Alpha (Binary)	CHAR(n)	VARBINARY(n)CHAR(n)
Auto sequence	BIGINT	BIGINT	BIGINT
Auto time	BIGINT	BIGINT	BIGINT
Binary	CHAR(n)	VARBINARY(n)CHAR(n)
Boolean	BIT	BOOLEAN	BOOLEAN
Date (YYYYMMDD)	DATE	DATE	DATE
Date (YYMMDD)	DATE	DATE	DATE
Date (YYYYJJJ)	DECIMAL(7)	DECIMAL(7)	NUMERIC(7)

Date (YYJJJ)	DECIMAL(5)	DECIMAL(5)	NUMERIC(5)
Date (YYYYPP)	DECIMAL(6)	DECIMAL(6)	NUMERIC(6)
Date (YYPP)	DECIMAL(4)	DECIMAL(4)	NUMERIC(4)
Decimal	DECIMAL(n)	DECIMAL(n)	NUMERIC(n)
Enum	INT	INT	INT
Implied Decimal	DECIMAL(n,p) _{DECIMAL(n,p)}	NUMERIC(n,p)
Integer (i1)	SMALLINT	TINYINT	SMALLINT
Integer (i2)	SMALLINT	SMALLINT	SMALLINT
Integer (i4)	INT	INT	INT
Integer (i8)	BIGINT	BIGINT	BIGINT
Structure field	VARCHAR(n)	VARCHAR(n)	VARCHAR(n)
Time (HHMMSS)	TIME(0)	TIME(0)	TIME(0)
Time (HHMM)	TIME(0)	TIME(0)	TIME(0)
User Defined Alpha	CHAR(n)	CHAR(n)	CHAR(n)
User Defined Numeric	CHAR(n)	CHAR(n)	CHAR(n)
User Defined Time-stamp (see below)	DATETIME2	TIMESTAMP	TIMESTAMP
User Defined Date (all others)	CHAR(n)	CHAR(n)	CHAR(n)

User-Defined Time-stamp Fields

User-defined time-stamp fields can be implemented in the same way as supported by xfODBC:

- Define the field as a user-defined date type, length 20
- Set the fields "user string" to ^CLASS^=YYYYMMDDHHMISSUUUUUU
- Use the <IF USERTIMESTAMP> and <IF NOTUSERTIMESTAMP> expressions to detect the fields in field loops.
- Use the SQL CONVERT function to convert a YYYYMMDDHHMISSUUUUUU string into a DATETIME2 value as required by SQL Server. Note that DATETIME2 fields are only supported from SQL Server 2008. An example of using CONVERT is shown below:

```
a20field = %datetime

a26field = %string(^d(a20field),"XXXX-XX-XX
XX:XX:XX:XXXXXXXX)
```

Then when building your SQL statement you would do something like this

```
sql = sql + "CONVERT(DATETIME2," + a26field + ",21)"
```

- <FIELD_CSTYPE>
- <FIELD_SPEC>
- <FIELD_TYPE>
- <FIELD_TYPENAME>
- <FIELD_VBTYPE>
- Other field loop expansion tokens

FIELD_TEMPLATE

Inserts the name of the Repository template from which the field inherited its attributes or by a null string if no template is referenced.

Usage

<FIELD_TEMPLATE>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD_TKSCRIPT

Inserts a UI Toolkit .field script file command for the field.

Usage

<FIELD_TKSCRIPT>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD TYPE

Inserts a character indicating the synergy data type of field.

Usage

<FIELD_TYPE>

Possible values

- A
- D
- I

Use with Implicit Groups

An implicit group is a group in the repository structure being processed whose fields are defined by another repository structure.

By default CodeGen will expand group fields to multiple individual fields based on the fields within the group that was encountered. As an alternative, if the -g i command line option is used (don't expand implicit groups to individual fields), and an implicit group is encountered, then the implicit group is assumed to be represented as a class, and the field type will be inserted as a class type. For example, for an implicit group based on the repository structure CUSTOMER, the field specification will be:

@CUSTOMER

Alternate forms

- <field_type>
- <Field_Type>
- <Field_type>

- <FieldType>
- <fieldType>

See also

- <FIELD_CSTYPE>
- <FIELD_OCTYPE>
- <FIELD_SPEC>
- <FIELD_SQLTYPE>
- <FIELD_TYPE_NAME>
- <FIELD_VBTYPE>
- Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_TYPE_NAME

Inserts a string indicating the Synergy data type of field.

Usage

<FIELD_TYPE_NAME>

Possible values

- ALPHA
- AUTO SEQUENCE
- AUTO TIME
- BINARY
- DATE
- DECIMAL
- IMPLIED
- INTEGER
- JULIAN
- TIME
- USER ALPHA
- USER DATE
- USER NUMERIC

Alternate forms

None

See also

- <FIELD_CSTYPE>
- <FIELD_SPEC>
- <FIELD_SQLTYPE>
- <FIELD_TYPE>
- <FIELD_VBTYPE>
- Other field loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FIELD_UTEXT

Inserts the fields' user-defined text string.

Usage

<FIELD_UTEXT>

Alternate forms

None

See also

• Other field loop expansion tokens

FIELD_VBDEFAULT

Inserts a VB.NET default value based on the fields data type.

Usage

<FIELD_VBDEFAULT>

Possible values

- """
- 0
- False
- New DateTime()
- Nothing

Alternate forms

None

- <FIELD_CSCONVERT>
- <FIELD_VBTYPE>
- Other field loop expansion tokens

FIELD_VBTYPE

Inserts a string indicating the VB.NET data type of the field.

Usage

<FIELD_VBTYPE>

Possible values

- Boolean
- Byte
- DateTime
- Decimal
- Integer
- Long
- Nullable(Of DateTime)
- Short
- String
- System.Double
- System.SByte
- System.Single
- System.UInt16
- System.UInt32

• System.UInt64

Alternate forms

None

See also

- <FIELD_CSTYPE>
- <FIELD_OCTYPE>
- <FIELD_SPEC>
- <FIELD_SQLTYPE>
- <FIELD_TYPE>
- <FIELD_TYPE_NAME>
- Other field loop expansion tokens

MAPPED FIELD

Inserts the name of the mapped field in the mapped structure. To use this token you must use @MAP=structure; in the structures user text field or long description and @MAP=field; in each fields user text field or long description.

Usage

<MAPPED_FIELD>

Alternate forms

- <mapped_field>
- <Mapped_Field>
- <Mapped_field>
- <MappedField>
- <mappedField>

- <MAPPED_PATH>
- <MAPPED_PATH_CONV>
- Other field loop expansion tokens
- Structure Mapping

MAPPED_PATH

Inserts the name of the mapped path in the mapped structure. To use this token you must use @MAP=structure; in the structures user text field or long description and @MAP=field; in each fields user text field or long description.

Usage

<MAPPED_PATH>

Alternate forms

<mapped_path>

See also

- <MAPPED_FIELD>
- <MAPPED_PATH_CONV>
- Other field loop expansion tokens
- Structure Mapping

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

MAPPED_PATH_CONV

Inserts the name of the mapped path in the mapped structure . To use this token you must use @MAP=structure; in the structures user text field or long description and @MAP=field; in each fields user text field. If there is also an @MAPF=function; in the user text or long description then that conversion function will also be applied.

Usage

<MAPPED_PATH_CONV>

Alternate forms

<mapped_path_conv>

See also

- <MAPPED_FIELD>
- <MAPPED_PATH>
- Other field loop expansion tokens
- Structure Mapping

PROMPT_COL

Inserts the screen column for placement of the prompt on the screen in a character based environment. If the prompt has a column position specified then this is used, otherwise the default of column 1 is used.

Usage

<PROMPT_COL>

Alternate forms

None

- <FIELD_COL>
- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_PIXEL_WIDTH>
- <FIELD_ROW>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_PIXEL_WIDTH>
- <PROMPT_ROW>
- Other field loop expansion tokens

PROMPT_PIXEL_COL

Inserts the pixel column for placement of the prompt on a windows form. If the prompt has a column position specified then this is used, otherwise the default of column 1 is used.

A default width of 12 pixels per character is used, and can be overridden using the -cw command line option.

Usage

<PROMPT_PIXEL_COL>

Alternate forms

None

- <FIELD_COL>
- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_PIXEL_WIDTH>
- <FIELD_ROW>
- <PROMPT_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_PIXEL_WIDTH>
- <PROMPT_ROW>



PROMPT_PIXEL_ROW

Inserts the pixel row for placement of the prompt on a windows form. If the prompt has a row position defined then this is used, otherwise the logical field number (which includes array elements) is used to calculate the default row.

A default height of 25 pixels per row is used, and can be overridden using the -ch command line option.

Usage

<PROMPT PIXEL ROW>

Alternate forms

None

- <FIELD_COL>
- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_PIXEL_WIDTH>
- <FIELD_ROW>
- <PROMPT_COL>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_WIDTH>
- <PROMPT_ROW>



PROMPT_PIXEL_WIDTH

Inserts the pixel width for default sizing of the prompt when represented as a label.

Calculated as the size of the prompt in characters multiplied by 12 pixels per character. This default character width can be overridden using the -cw command line option.

Usage

<PROMPT_PIXEL_WIDTH>

Alternate forms

None

- <FIELD_COL>
- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_PIXEL_WIDTH>
- <FIELD_ROW>
- <PROMPT_COL>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_ROW>



PROMPT_ROW

Inserts the screen row for placement of the prompt on the screen in a character based environment. If the prompt has a row position defined then this is used, otherwise the logical field number (which includes array elements) is used to calculate the default row.

Usage

<PROMPT_ROW>

Alternate forms

None

- <FIELD_COL>
- <FIELD_DRILL_PIXEL_COL>
- <FIELD_PIXEL_COL>
- <FIELD_PIXEL_ROW>
- <FIELD_PIXEL_WIDTH>
- <FIELD_ROW>
- <PROMPT_COL>
- <PROMPT_PIXEL_COL>
- <PROMPT_PIXEL_ROW>
- <PROMPT_PIXEL_WIDTH>
- Other field loop expansion tokens

Field Loop Expression Tokens

Field loop expressions are expressions that can only be used within the context of a field loop and allow you to conditionally include or exclude code based on the attributes of the current field.

Field Loop Expression	Description
<if alpha=""></if>	Data type is Alpha.
<if allow_list=""></if>	The field has allow list values.
<if alternate_name=""></if>	The field has an alternate name specified. The <field_altname> token will always return a value because if no alternate name is specified in the repository then it defaults to the fields real name. This token allows you to detect whether the alternate name is different from the real name.</field_altname>
<if array=""></if>	The field used to be part of an array with 1, 2, 3 or 4 dimensions.
<if array_first=""></if>	The field used to be the very first element of an array.
<if array1=""></if>	The field used to be part of an array with 1 dimension.
<if array1_first=""></if>	The field used to be the very first element of an array with 1 dimension.
<if array2=""></if>	The field used to be part of an array with 2 dimensions.
<if array2_first=""></if>	The field used to be the very first element of an array with 2 dimensions.

<if array3=""></if>	The field used to be part of an array with 3 dimensions.
<if array3_first=""></if>	The field used to be the very first element of an array with 3 dimensions.
<if array4=""></if>	The field used to be part of an array with 4 dimensions.
<if array4_first=""></if>	The field used to be the very first element of an array with 4 dimensions.
<if arrive=""></if>	The field has an arrive method specified.
<if auto_sequence=""></if>	The field is an auto-sequence field. The underlying storage is an I8.
<if auto_timestamp=""></if>	The field is an auto-time-stamp field. This expression will evaluate to true if the repository field is defined as type AutoTime (AT). The underlying storage is an I8.
<if AUTO_TIMESTAMP_CREATED></if 	The field is an auto-time-stamp field AND is associated with a key segment
<if auto_timestamp_updated=""></if>	The field is an auto-time-stamp field AND is associated with a key segment that is defined as type AutoTime Created (C). The underlying storage is an I8.
<if binary=""></if>	The field is a binary field. Binary fields are also alpha fields.
<if bold=""></if>	The field should be displayed in bold.
<if boolean=""></if>	The fields data type is boolean. This only applies to fields whose actual Synergy data type is boolean, not to numeric fields with a coerced type of boolean.

<if bzero=""></if>	The numeric field should be displayed blank if the field value is zero.
<if break=""></if>	The field is a break field. This expression matches any type of break condition (change, always or return).
<if break_always=""></if>	The field is a break field. The break field logic always executes whenever the field is processed.
<if break_change=""></if>	The field is a break field. The break field logic only executes when the value of the field changes.
<if break_return=""></if>	The field is a break field. The break field logic only executes when the user presses RETURN on the field.
<if change=""></if>	The field has a change method specified.
<if checkbox=""></if>	The field should be displayed as a checkbox.
<if coerce_boolean=""></if>	The field data type is coerced to boolean.
<if combobox=""></if>	The field should be displayed as a combo-box.
<if date=""></if>	The field is a date or nullable date.
<if dateortime=""></if>	The field is a date, nullable date, or time.
<if datetoday=""></if>	The date field should default to the current date.
<if date_julian=""></if>	The field is a date field of type YYJJJ or YYYYJJJ.
	The field is a date field but is NOT a

<if date_not_julian=""></if>	Julian date. It could be a reverse date or a period date.
<if date_not_nullable=""></if>	The field is a date, but not a nullable date.
<if date_not_period=""></if>	The field is a date field but is NOT a period date. It could be a reverse date or a Julian date.
<if date_not_ymd=""></if>	The field is a date field but is NOT a YYMMDD or YYYYMMDD date.
<pre><if date_not_yyyymmdd=""></if></pre>	The field is a date field but is NOT a YYYYMMDD date.
<if date_nullable=""></if>	The field is a NULLABLE date.
<if date_period=""></if>	The field is a date field of type YYPP or YYYYPP.
<if date_ymd=""></if>	The field is a date field of type YYMMDD or YYYYMMDD.
<if date_yymmdd=""></if>	The field is a date field of type YYMMDD.
<if date_yyyymmdd=""></if>	The field is a date field of type YYYYMMDD.
<if date_yyjjj=""></if>	The field is a date field of type YYJJJ.
<if date_yyyyjjj=""></if>	The field is a date field of type YYYYJJJ.
<if date_yypp=""></if>	The field is a date field of type YYPP.
<if date_yyyypp=""></if>	The field is a date field of type YYYYPP.

<if decimal=""></if>	The field data type is decimal. Note that date, nullable date and time fields are NOT treated as decimal fields with this conditional test.
<if default=""></if>	The field has a default value specified.
<if description=""></if>	The field has a description specified.
<if disabled=""></if>	The field is defined as being initially disabled.
<if display=""></if>	The field has a display method specified.
<if display_length=""></if>	The field has a display length specified.
<if drill=""></if>	The field has a drill method specified.
<if echo=""></if>	The field should echo characters as they are typed.
<if editformat=""></if>	The field has an edit format method specified.
<if enabled=""></if>	The field is defined as being initially enabled.
<if enum=""></if>	The field is an ENUM field. This is a real language enumeration, not a UI Toolkit enumerated field.
<if enumerated=""></if>	The field has a UI Toolkit enumeration specified. This is not the same as being a real language ENUM field.
<if field_position=""></if>	The field has a UI field position specified.
<if format=""></if>	The field has a display format specified.

<if group_expand=""></if>	Implicit groups are being expanded to individual fields (i.e. the -g i command line option is not being used).
<if group_no_expand=""></if>	Implicit groups are not being expanded to individual fields (i.e. the -g i command line option is being used).
<if heading=""></if>	The field has a report heading specified.
<if helpid=""></if>	The field has a help identifier specified.
<if hyperlink=""></if>	The field has a hyperlink method specified.
<if i1=""></if>	The field is an I1.
<if i2=""></if>	The field is an I2.
<if i4=""></if>	The field is an I4.
<if i8=""></if>	The field is an I8.
<if i124=""></if>	The field is an I1, I2 or I4.
<if infoline=""></if>	The field has information line text specified.
<if input_center=""></if>	Input should be performed center justified.
<if input_left=""></if>	Input should be performed left justified.
<if input_right=""></if>	Input should be performed right justified.
<if integer=""></if>	The field data type is defined as integer.
<if language=""></if>	The field is available to the language.

<if leave=""></if>	The field has a leave method specified.
<if length_over_8=""> <if longdesc=""></if></if>	The length of the field is over 8 characters. This expression is primarily useful when dealing with numeric fields because decimal fields with a length of up to 8 digits are mapped to an int, whereas fields 9 characters or longer are mapped to a long. The field has a long description specified.
<if mapped=""></if>	The field has a CodeGen field mapping specified.
<if mappedstr=""></if>	The fields' parent structure has a CodeGen structure mapping specified.
<if negative_allowed=""></if>	The field allows negative values. Use of this expression is only appropriate for numeric fields.
<if negative_orzero=""></if>	The field allows negative values or a value of zero. Positive values are not allowed. Use of this expression is only appropriate for numeric fields.
<if negative_required=""></if>	The field requires a negative value. Use of this expression is only appropriate for numeric fields.
<if noallow_list=""></if>	The field does not have allow list values specified.
<if noalternate_name=""></if>	The field does not have an alternate name specified. The <field_altname> token will always return a value because if no alternate name is specified in the repository then it defaults to the fields</field_altname>

real name. This token allows you to
detect whether the alternate name is the
same as the real name.

	detect whether the alternate name is the same as the real name.
<if noarrive=""></if>	The field does not have an arrive method.
<if nobreak=""></if>	The field id not a break field.
<if nochange=""></if>	The field does not have a change method.
<if nocheckbox=""></if>	The field should not be displayed as a check-box.
<if nocoerceboolean=""></if>	The field data type is not coerced to boolean.
<if nodefault=""></if>	The field does not have a default value.
<if nodescription=""></if>	The field does not have a description.
<if nodisplay=""></if>	The field does not have a display method.
<if nodisplay_length=""></if>	The field does not have a display length.
<if nodrill=""></if>	The field does not have a drill method.
<if noecho=""></if>	The field should not echo characters as they are typed.
<if noeditformat=""></if>	The field does not have an edit format method.
<if noformat=""></if>	The field does not have a format string.
<if nohelpid=""></if>	The field does not have a help ID.
<if nohyperlink=""></if>	The field does not have a hyperlink method.

<if noinfoline=""></if>	The field does not have information line text.
<if nolanguage=""></if>	The field is marked as "Excluded by Language".
<if noleave=""></if>	The field does not have a leave method.
<if nolongdesc=""></if>	The field does not have a long description specified.
<if nonegative=""></if>	The field does not allow negative values. This expression should only be used with numeric fields.
<if nopaintchar=""></if>	The field does not have a custom paint character.
<if noprecision=""></if>	The field does not have a decimal precision specified (i.e. it is not an implied decimal field).
<if noprompt=""></if>	The field does not have a prompt.
<if norange=""></if>	The field does not have a numeric range specified. This expression should only be used for numeric fields.
<if noreport=""></if>	The field is marked as "Excluded by ReportWriter".
<if noselections=""></if>	The field does not have selection list values.
<if noselwnd=""></if>	The field does not have a selection window name specified.
<if notalpha=""></if>	The field data type is not defines as Alpha.

<if notarray=""></if>	The field was not part of an array.
<if notbinary=""></if>	The field is not a binary field. Binary fields are also alpha fields.
<if notboolean=""></if>	The field is not a boolean field.
<if notbzero=""></if>	The numeric field should not be displayed as blank if the value is zero.
<if notdate=""></if>	The field is not a date or nullable date.
<if notdateortime=""></if>	The field is not a date, nullable date or time.
<if notdatetoday=""></if>	The date field should not default to the current date.
<if notdecimal=""></if>	The field data type is not defined as decimal. Note that date, nullable date and time fields do not count as decimal with this conditional test.
<if notenum=""></if>	The field is not a language ENUM field.
<if notenumerated=""></if>	The field is not a UI Toolkit enumerated field.
<if notimeout=""></if>	The field does not have an input timeout specified.
<if notinteger=""></if>	The field data type is not defined as integer.
<if notnumeric=""></if>	The field data type is not a numeric type.
<if notoverlay=""></if>	The field is not an overlay field or overlay group.
	The field is not a segment in the primary

<if notpksegment=""></if>	key.
<if notradiobuttons=""></if>	The field should not be viewed as a collection of radio-buttons.
<if notstructfield=""></if>	The field is not a struct field.
<if notoolkit=""></if>	The field is marked as "Excluded by Toolkit".
<if notuppercase=""></if>	The field should not be converted to uppercase. Only appropriate for alpha fields.
<if notuser=""></if>	The field is not a user-defined field type.
<if nousertext=""></if>	The field does not have user text specified.
<if noview_length=""></if>	The field does not have a view length specified.
<if nottime=""></if>	The field is not marked as a time field.
<if notusertimestamp=""></if>	The field is not a user-defined timestamp field. (See <field_sqltype> for more information).</field_sqltype>
<if noweb=""></if>	The field is marked as "Excluded by Web".
<if numeric=""></if>	The field data type is defines as a numeric type.
<if ocnative=""></if>	The fields Objective-C data type is a native type (e.g. int or BOOL).
<if ocobject=""></if>	The fields Objective-C data type is an object type (e.g. NSString or NSDate).

<if optional=""></if>	The field is marked as optional (i.e. not required).
<if overlay=""></if>	The field is an overlay field or overlay group.
<if paintchar=""></if>	The field has a custom paint character specified.
<if pksegment=""></if>	The field is a segment in the primary key.
<if precision=""></if>	The decimal field has a precision specified (i.e. it is an implied decimal field).
<if prompt=""></if>	The field has a prompt specified.
<if prompt_position=""></if>	The field has a UI prompt position specified.
<if radiobuttons=""></if>	The field should be viewed as a collection of radio-buttons.
<if range=""></if>	The field has a numeric range (minimum/maximum values) specified.
<if readonly=""></if>	The field state is specified as initially read-only.
<if readwrite=""></if>	The field state is specified as initially writable.
<if report=""></if>	The field is available to ReportWriter.
<if report_center=""></if>	The field should be printed center justified on reports.
<if report_left=""></if>	The field should be printed left justified

on reports.

<IF REPORT_RIGHT>
The field should be printed right justified

on reports.

<IF REQUIRED> The field is marked as required (i.e. must

contain a valid value).

<IF REVERSE> The field should be displayed in reverse

video.

<IF SELECTIONS>
The field has one or more selection list

values.

<IF SELWND> The field has a selection window name

specified.

<IF STRUCTFIELD> The field is a struct field.

<IF TEXTBOX>
The field should be displayed as a text-

box.

<IF TIME> The field is a time field of type HHMM

or HHMMSS.

<IF TIME_HHMM> The field is a time field of type HHMM.

<IF TIME_HHMMSS>
The field is a time field of type

HHMMSS.

<IF TIMEOUT>
The field has an input timeout specified.

<IF TIMENOW>
The time field should default to the

current time.

<IF TOOLKIT>
The field is available to UI Toolkit.

<IF UNDERLINE>
The field content should be displayed

underlined.

<if uppercase=""></if>	The field value should always be

converted to uppercase.

The field is a user defines field type. <IF USER>

<IF USERTEXT> The field has user text specified.

<IF VIEW_LENGTH> The field has a view length specified.

The field is a user-defined time-stamp <IF USERTIMESTAMP>

field. (See <FIELD_SQLTYPE> for

more information).

The field is available to the web <IF WEB>

products.

Field Selection Loops

A field selection loop is a template file construct which allows you to iterate through the collection of selection values that are associated with a repository field definition. In order to use a field selection loop you must be processing a repository structure, either directly via the —s command line option, or because the structure is referenced by the UI Toolkit input window that you are processing via the —w command line option.

Field selection loops are delimited by a matching pair of <SELECTION_LOOP> and </SELECTION_LOOP> tags, which surround the template code to be inserted for each selection list.

Field selection loop tokens are tokens that can only be used within a field selection loop:

- Field Selection Loop Expansion Tokens
- Field Selection Loop Expression Tokens

There are two types of field selection loop.

Multi-line Field Selection Loops

Multi-line field selection loops occur when the opening and closing loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
[code]

<SELECTION_LOOP>code
</SELECTION_LOOP>
[code]
```

Multi-line field selection loops generate one or more lines of output code for each key that is processed.

In-line Field Selection Loops

In-line field selection loops exist when the opening and closing loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

[code] **SELECTION_LOOP**> code **SELECTION_LOOP**> [code]

In-line field selection loops generate code into the current output line only.

The code between the opening and closing tags of the loop is repeated for each field selection list entry in the field being processed.

The code within a field selection loop can contain generic tokens, structure tokens and special field selection loop tokens, which are discussed later. Most field loop tokens can also be used inside a field selection loop.

Field selection loop tokens can only be used inside a field selection loop, and a field selection loop can only be declared within the context of a field loop.

Field Selection Loop Expansion Tokens

Field selection loop expansion tokens are used to insert information about the current field selection value being processed within the context of a field selection loop into the output stream. Field selection loop tokens can only be used inside a field selection loop.

Field Selection Loop Token Description

<selection_count></selection_count>	Inserts the number of selections associated with the field.
<selection_nuimber></selection_nuimber>	Inserts the one-based index of the current selection within the selections collection.
<selection_text></selection_text>	Inserts the text associated with the current selection.
<selection_value></selection_value>	Inserts the value associated with the current selection.

SELECTION_COUNT

Inserts the number of selections associated with the field.

Usage

<SELECTION_COUNT>

Alternate forms

None

See Also

None

SELECTION_NUMBER

Inserts the one-based index of the current selection within the selections collection.

Usage

<SELECTION_NUMBER>

Alternate forms

None

See Also

None

SELECTION_TEXT

Inserts the text associated with the current selection.

Usage

<SELECTION_TEXT>

Alternate forms

None

See Also

None

SELECTION_VALUE

Inserts the value associated with the current selection.

For numeric fields the numeric value is calculated according to the fields enumeration properties (base value and step value). If the field does not have enumeration properties specified then the default base value is 1 and the default step value is 1.

For alpha fields the value text is deduced from the left portion of the display text, based on the size of the field. For example, if an A2 field has a selection item of "Credit Card", the selection value is "Cr".

Usage

<SELECTION_VALUE>

Alternate forms

None

See Also

None

Field Selection Loop Expression Tokens

CodeGen includes support for field selection loop expression tokens, but none have been implemented yet. Loop utility expressions can also be used within field selection loops.

Key Loops

A key loop is a template file construct which allows you to iterate through the collection of keys that CodeGen has information about. In order to use a key loop you must be processing a repository structure, either directly via the —s command line option, or because the structure is referenced by the UI Toolkit input window that you are processing via the —w command line option.

Note: The Synergy/DE Repository does not have an adequate mechanism to ensure that repository key definitions match the actual key definitions of the ISAM files being represented. Because of this, CodeGen considers the primary key to be the first key defined in the repository, and assumes that the alternate keys will immediately follow the primary key definition, and be in the correct sequence, as defined in the ISAM file. Any foreign keys should be defined after all of the access key definitions.

Key loops are delimited by a matching pair of **KEY_LOOP**> and **KEY_LOOP**> tags that surround the template code to be inserted for each key.

Key loop tokens are tokens that can only be used within a key loop:

- Key Loop Expansion Tokens
- Key Loop Expression Tokens

There are two types of key loops, multi-line key loops and in-line key loops.

Multi-Line Key Loops

Multi-line key loops occur when the opening and closing key loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
[code]

<KEY_LOOP>code

</KEY_LOOP>
```

[code]

Multi-line key loops generate one or more lines of output code for each key that is processed.

In-Line Key Loops

In-line key loops exist when the opening and closing key loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
[code] KEY_LOOP code KEY_LOOP [code]
```

In-line key loops generate code into the current output line only.

The code between the opening and closing tags of a key loop is repeated for each key in the structure being processed.

This code within a key loop can contain other generic and structure tokens, and can also contain special key loop tokens, which are discussed later.

Key loop tokens can only be used inside a key loop, and a key loop can't be declared within any other loop construct.

There are two additional variations on a key loop. These variations are called alternate key loops, and primary key blocks.

Alternate Key Loops

An alternate key loop is similar to a key loop; except that only alternate keys are processed (i.e. the primary key is skipped). As with key loops, CodeGen supports both multi-line and in-line alternate key loops.

A multi-line alternate key loop looks like this:

[code]

```
<aLTERNATE_KEY_LOOP>
code
</ALTERNATE_KEY_LOOP>
```

[code]

And an in-line alternate key loop looks like this:

```
[code] ALTERNATE_KEY_LOOP> code ALTERNATE_KEY_LOOP> [code]
```

Multi-line alternate key loops generate one or more lines of output code for each key that is processed, whereas in-line alternate key loops generate code into the current output line only.

Primary Key Blocks

A primary key block is similar to a key loop; except that only the primary key is processed (i.e. all alternate keys are skipped). As with key loops, CodeGen supports both multi-line and in-line primary key blocks.

A multi-line primary key block looks like this:

```
[code]
<PRIMARY_KEY>
code
</PRIMARY_KEY>
[code]
```

And an in-line primary key block looks like this:

```
[code] PRIMARY_KEY> code PRIMARY_KEY> [code]
```

Multi-line primary key blocks generate one or more lines of output code as the primary key is processed, whereas in-line primary key blocks generate code into the current output line only.

Unique Key Blocks

A unique key block is similar to a key loop; except that only the first unique key (i.e. the first key that does not allow duplicate key values) is processed. This will usually be the primary key, but could be some alternate key in the rare case that the primary key allows duplicates. As with key loops, CodeGen

supports both multi-line and in-line unique key blocks.

A multi-line unique key block looks like this:

```
[code]
<UNIQUE_KEY>
code
</UNIQUE_KEY>
[code]
```

And an in-line unique key block looks like this:

```
[code] <UNIQUE_KEY> code </UNIQUE_KEY> [code]
```

Multi-line unique key blocks generate one or more lines of output code as the first unique key is processed, whereas in-line unique key blocks generate code into the current output line only.

If you attempt to use a unique key block in conjunction with a repository structure that does not have any unique keys then an error will be generated. You can avoid this by isolating the unique key block using an <IF STRUCTURE_HAS_UNIQUE_KEY> expression.

Key Loop Example

Template code like this:

```
NAME
                            "<KEY_NAME>"
                        <IF DUPLICATES>yes</IF><IF
    DUPLICATES
NODUPLICATES>no</IF>
    <IF DUPLICATES>
    DUPLICATE_ORDER
                          <IF
DUPLICATESATFRONT>lifo</IF><IF
DUPLICATESATEND>fifo</IF>
    </IF>
                         <IF CHANGES>yes</IF><IF
    MODIFIABLE
NOCHANGES>no</IF>
    <IF NULLKEY>
                            <key_nulltype>
    NULL
    <IF NULLVALUE>
    VALUE_NULL
                         <KEY_NULLVALUE>
    </IF>
    </IF>
                          <KEY_DENSITY>
    DENSITY
</KEY_LOOP>
```

Would process each of a structures defined access keys, and generate output code like this:

KEY 0

START	23
LENGTH	8
TYPE	alpha
ORDER	ascending
NAME	"PROJECT_KEY0"
DUPLICATES	no
MODIFIABLE	no
DENSITY	50

KEY 1

START	31:41:51
LENGTH	10:10:3

TYPE alpha:alpha

ORDER ascending:ascending

NAME "PROJECT_KEY1"

DUPLICATES no MODIFIABLE no DENSITY 50

KEY 2

START 114 LENGTH 8

TYPE alpha ORDER ascending

NAME "PROJECT_KEY2"

DUPLICATES yes
DUPLICATE_ORDER fifo
MODIFIABLE yes
DENSITY 50

KEY3

START 140:130 LENGTH 15:2

TYPE alpha:alpha
ORDER ascending:ascending

NAME "PROJECT_KEY3"

DUPLICATESyesDUPLICATE_ORDERfifoMODIFIABLEyesDENSITY50

KEY 4

START 934 LENGTH 20

TYPE alpha ORDER ascending

NAME "REPLICATION_KEY"

```
DUPLICATES no
MODIFIABLE yes
DENSITY 50
```

Alternate Key Loop Example

Template file code like this:

```
<ALTERNATE_KEY_LOOP>
;;Create index <KEY NUMBER> (<KEY DESCRIPTION>)
if (ok)
begin
 sql = "CREATE <KEY_UNIQUE> INDEX
IX_<STRUCTURE_NAME>_<KEY_NAME> "
     "ON <STRUCTURE NAME>(<SEGMENT LOOP>
<SEGMENT_NAME> <SEGMENT ORDER><,>
</SEGMENT LOOP>)"
 call open_cursor
 if (ok)
 begin
   call execute cursor
   call close cursor
 end
end
</ALTERNATE_KEY_LOOP>
```

Would process each of a structures defined access keys, except for the primary key, and generate output code like this:

```
;;-----;;Create index 1 (Projects by customer)
```

```
if (ok)
begin
    sql = "CREATE UNIQUE INDEX
IX_PROJECT_PROJECT_KEY1 "
        "ON PROJECT(CUSTOMER_ID ASC,CONTRACT_ID
ASC,CONTRACT_PROJECT_ID ASC)"
    call open_cursor
    if (ok)
    begin
      call execute_cursor
      call close_cursor
    end
end
;;Create index 2 (Projects by start date)
;;
if (ok)
begin
    sql = "CREATE INDEX IX_PROJECT_PROJECT_KEY2"
        "ON PROJECT(START_DATE ASC)"
    call open_cursor
    if (ok)
    begin
      call execute_cursor
      call close_cursor
    end
end
;;Create index 3 (Projects by consultant and status)
;;
```

```
if (ok)
begin
    sql = "CREATE INDEX IX_PROJECT_PROJECT_KEY3"
        "ON PROJECT(LEAD_CONSULTANT
ASC, CURRENT_STATUS ASC)"
    call open_cursor
    if (ok)
    begin
      call execute_cursor
      call close_cursor
    end
end
;;Create index 4 (SQL Timestamp Key)
;;
if (ok)
begin
    sql = "CREATE UNIQUE INDEX
IX PROJECT REPLICATION KEY"
        "ON PROJECT(REPLICATION_KEY ASC)"
    call open_cursor
    if (ok)
    begin
      call execute cursor
      call close cursor
    end
end
```

Primary Key Block Example

Template file code like this:

```
<PRIMARY KEY>
<SEGMENT LOOP>
```

```
{xfParameter(name="<SegmentName>")}
required in a<SegmentName>, <segment_spec>
</SEGMENT_LOOP>
</PRIMARY_KEY>

{xfParameter(name="
<StructureName>",collectionType="structure",structure="
<STRUCTURE_NAME>",dataTable="true")}
required out a<StructureName>s, @ArrayList
```

When processed for a structure with only a single segment in it's primary key, would produce output like this

```
{xfParameter(name="ProjectId")}
required in aProjectId, d8

{xfParameter(name="Project",collectionType="structure",structure="PRC required out aProjects, @ArrayList

When processed for a structure with multiple segments in its primary key, would produce output like this:

{xfParameter(name="TaskId")}
```

```
required in aTaskId, d3
{xfParameter(name="AttachmentId")}
required in aAttachmentId, d3
```

{xfParameter(name="ProjectAttachment",collectionType="structure",stru required out aProjectAttachments, @ArrayList

Key Loop Expansion Tokens

Key loop expansion tokens are used to insert information about the current key being processed within the context of a key loop into the output stream. Key loop tokens can only be used inside a key loop.

Key Loop Token	Description
<key_changes></key_changes>	Inserts text indicating whether the key allows changes to existing key entries.
<key_density></key_density>	Inserts the key packing density for the current key.
<key_description></key_description>	Inserts the description of the key. Be aware that key descriptions are optional in Repository, so
<key_duplicates></key_duplicates>	this token may expand to nothing! Inserts text indicating whether the key allows duplicate values.
<key_duplicates_at></key_duplicates_at>	Inserts text indicating where duplicate values are inserted.
<key_length></key_length>	Inserts the length of the key. The length of the key is the sum of the length of all key segments.
<key_name></key_name>	Inserts the name of the key. Note that this is the name of the key in the repository, and may or may not correspond to the name of the key in the underlying ISAM file, if a key name exists.
<key_nulltype></key_nulltype>	Inserts the keys null key type. The token should only be used within a <if nullkey=""> expression.</if>
<key_nullvalue></key_nullvalue>	Inserts the keys null key value. The token should only be used within an <if nullkey=""> expression.</if>
<key_number></key_number>	Inserts the number of the key, primary key is key 0, first alternate key is key 1, and so on.
<key_order></key_order>	Inserts the ordering sequence of the key, ascending or descending.
<key_segments></key_segments>	Inserts the number of key segments defined.



If the key does not allow duplicates then inserts the text UNIQUE, otherwise nothing is inserted.

KEY_CHANGES

Inserts text indicating whether the key allows changes to existing key entries.

Usage

<KEY_CHANGES>

Possible values

- CHANGES
- NOCHANGES

Alternate forms

None

See Also

None

KEY_DENSITY

Inserts the key packing density for the current key.

Usage

<KEY_DENSITY>

Possible values

• A number in the range 50 to 100.

Alternate forms

None

See Also

None

KEY_DESCRIPTION

Inserts the description of the key. Be aware that key descriptions are optional in Repository, so this token may expand to nothing!

Usage

<KEY_DESCRIPTION>

Alternate forms

None

See Also

None

KEY_DUPLICATES

Inserts text indicating whether the key allows duplicate values.

Usage

<KEY_DUPLICATES>

Possible values

- DUPLICATES
- UNIQUE

Alternate forms

None

See Also

None

KEY_DUPLICATES_AT

Inserts text indicating where duplicate values are inserted.

Usage

<KEY_DUPLICATES_AT>

Possible values

- FRONT
- END

Alternate forms

None

See Also

None

KEY_LENGTH

Inserts the length of the key. The length of the key is the sum of the length of all key segments.

Usage

<KEY_LENGTH>

Alternate forms

None

See Also

None

KEY_NAME

Inserts the name of the key. Note that this is the name of the key in the repository, and may or may not correspond to the name of the key in the underlying ISAM file, if a key name exists.

Usage

<KEY_NAME>

Alternate forms

- <key_name>
- <Key_Name>
- <Key_name>
- <KeyName>
- <keyName>

See Also

None

KEY_NULLTYPE

Inserts the keys null key type. The token should only be used within a <IF NULLKEY> expression.

Usage

<KEY_NULLTYPE>

Possible values

- NONE
- REPLICATE
- NOREPLICATE
- SHORT

Alternate forms

<key_nulltype>

See Also

None

KEY_NULLVALUE

Inserts the keys null key value. The token should only be used within an <IF NULLKEY> expression.

Usage

<KEY_NULLVALUE>

Alternate forms

None

See Also

None

KEY_NUMBER

Inserts the number of the key, primary key is key 0, first alternate key is key 1, and so on.

Usage

<KEY_NUMBER>

Possible values

• A number in the range 0 to 1023.

Alternate forms

None

See Also

None

KEY_ORDER

Inserts the ordering sequence of the key, ascending or descending.

Usage

<KEY_ORDER>

Possible values

- ASC
- DESC

Alternate forms

None

See Also

None

KEY_SEGMENTS

Inserts the number of key segments defined.

Usage

<KEY_SEGMENTS>

Possible values

• A number in the range 1 to 8.

Alternate forms

None

See Also

None

KEY_UNIQUE

If the key does not allow duplicates then inserts the text UNIQUE, otherwise nothing is inserted.

Usage

<KEY_UNIQUE>

Alternate forms

None

See Also

None

Key Loop Expression Tokens

Key loop expressions are expressions that can only be used within the context of a key loop and allow you to conditionally include or exclude code based on the attributes of the current key.

Key Loop Expression	Description
<if ascending=""></if>	The key is ordered in ascending sequence.
<if auto_sequence=""></if>	The first segment of the key is an auto-sequence segment (ISAM Rev 6).
<if auto_timestamp=""></if>	The first segment of the key is an auto- time-stamp key, of one of the two types shown below. ISAM Rev 6 (Synergy/DE 10.1.1 or higher).
<if AUTO_TIMESTAMP_CREATED></if 	The first segment of the key is an auto- time-stamp key with values that are set only when records are first created. ISAM Rev 6 (Synergy/DE 10.1.1 or higher).
<if AUTO_TIMESTAMP_UPDATED></if 	The first segment of the key is an auto-
<if changes=""></if>	The key value may be changed.
<if descending=""></if>	The key is ordered in descending sequence.
<if duplicates=""></if>	The key allows records with duplicate key values.
<if duplicatesatend=""></if>	Duplicate key values are inserted after any existing records with the same key value.

<if duplicatesatfront=""></if>	Duplicate key values are inserted before any existing records with the same key value.
<pre><if first_seg_nocase=""> <if first_unique_key=""> <if multiple_segments=""></if></if></if></pre>	The first segment of the key is a non-case sensitive alpha field. The key being processed is the first unique key in the file. The key has more than one segment.
<if nochanges=""></if>	The key value may not be changed.
<if noduplicates=""></if>	The key does not allow multiple records with the same key value.
<if nullkey=""></if>	The key is a null key. See <key_nulltype> and <key_nullvalue>.</key_nullvalue></key_nulltype>
<if nullvalue=""></if>	The key has a null key value specified. See <key_nulltype> and <key_nullvalue>.</key_nullvalue></key_nulltype>
<if single_segment=""></if>	The key only has one segment.

Key Segment Loops

A key segment loop is a template file construct which allows you to iterate through the collection of key segments for a particular key that CodeGen has information about. In order to use a key segment loop you must be processing a repository structure, either directly via the —s command line option, or because the structure is referenced by the UI Toolkit input window that you are processing via the —w command line option, and must be inside a key loop.

Key segment loops are delimited by a matching pair of <SEGMENT_LOOP> and </SEGMENT_LOOP> tags, which surround the template code to be inserted for each key segment.

Key segment loop tokens are tokens that can only be used within a key segment loop:

- Key Segment Loop Expansion Tokens
- Key Segment Loop Expression Tokens

There are two types of key segment loop, multi-line key segment loops and inline key segment loops.

Multi-Line Key Segment Loops

Multi-line key segment loops occur when the opening and closing key segment loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
<KEY_LOOP>[code]
<SEGMENT_LOOP>
code
</SEGMENT_LOOP>
[code]
</KEY_LOOP>
```

Multi-line key segment loops generate one or more lines of output code for each key segment that is processed.

In-Line Key Segment Loops

In-line key segment loops exist when the opening and closing key segment loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
<KEY_LOOP>
[code]
[code] <SEGMENT_LOOP> code </SEGMENT_LOOP> [code]
[code]
</KEY_LOOP>
```

In-line key loops generate code into the current output line only.

Note: Key segment loops can also be used within alternate key loops and primary key blocks. Also, in the last example, the outer key loop could also be an in-line key loop.

The code between the opening and closing tags of a key segment loop is repeated for each key segment in the key being processed.

This code within a key segment loop can contain generic tokens, structure tokens, and special key segment loop tokens, which are discussed later. Most key loop tokens can also be used inside a key segment loop.

Key segment loop tokens can only be used inside a key segment loop, and a key segment loop can only be declared within one of the supported key loop constructs.

Key Segment Filter Loops

A key segment filter loop is similar to a key segment loop; except that the final key segment is not processed. As with key segment loops, CodeGen supports both multi-line and in-line key segment filter loops.

A multi-line key segment filter loop looks like this:

```
<KEY_LOOP>
[code]
<SEGMENT_LOOP_FILTER>
```

```
code
</SEGMENT_LOOP_FILTER>
[code]
</KEY_LOOP>
```

And an in-line key segment filter loop looks like this:

```
<KEY_LOOP>
[code]
[code] <SEGMENT_LOOP_FILTER> code
</SEGMENT_LOOP_FILTER> [code]
[code]
</KEY_LOOP>
```

Note: Key segment filter loops can also be used within alternate key loops and primary key blocks. Also, in the last example, the outer key loop could also be an in-line key loop.

If you use a segment filter loop with a key which only has one segment, then no code will be inserted into the output file.

First Segment Restrictions

A first segment restriction behaves in a similar way to a key segment loop; except that only the first key segment is processed. As with key segment loops, CodeGen supports both multi-line and in-line first segment restrictions.

A multi-line first segment restriction looks like this:

```
<KEY_LOOP>
[code]
<FIRST_SEGMENT>
code
</FIRST_SEGMENT>
[code]
</KEY_LOOP>
```

And an in-line first segment restriction looks like this:

```
<KEY_LOOP>
```

```
[code]
[code] <FIRST_SEGMENT> code </FIRST_SEGMENT> [code]
[code]
</KEY_LOOP>
```

Note: first segment restrictions can also be used within alternate key loops and primary key blocks. Also, in the last example, the outer key loop could also be an in-line key loop.

Second Segment Restrictions

Similar to a first segment restriction, as second segment restriction can be used to indicate that only the second segment of a key should be processed within a key loop. This can be particularly useful when processing structures that have a structure tag value as the first key segment and the main useful segment of the key is often then the second segment.

A multi-line second segment restriction looks like this:

```
<KEY_LOOP>
[code]
<SECOND_SEGMENT>
code
</SECOND_SEGMENT>
[code]
</KEY_LOOP>
```

And an in-line first segment restriction looks like this:

```
<KEY_LOOP>
[code]
[code] <SECOND_SEGMENT> code
</SECOND_SEGMENT> [code]
[code]
</KEY_LOOP>
```

Key Segment Loop Expansion Tokens

Key segment loop expansion tokens are used to insert information about the current key segment being processed within the context of a key segment loop into the output stream. Key segment loop tokens can only be used inside a key segment loop.

Key Segment Loop Token	Description
<segment_cstype></segment_cstype>	Inserts C# data type of the field associated with the current key segment.
<segment_desc></segment_desc>	Inserts the description of the field associated with the current key segment (if segment kind is field).
<segment_idxtype></segment_idxtype>	Inserts the index type of the key segment.
<segment_kind></segment_kind>	Inserts a numeric code for the kind of the current key segment.
<segment_length></segment_length>	Inserts the length of the key segment.
<segment_litval></segment_litval>	Inserts the literal value of the key segment (if segment kind is "literal").
<segment_mappedname></segment_mappedname>	Inerts the name of the mapped field corresponding to the current key segment (if segment kind is field).
<segment_name></segment_name>	Inserts the name of the field associated with the current key segment (if segment kind is field).
<segment_number></segment_number>	Inserts the one-based key segment number.
<segment_order></segment_order>	Inserts the ordering sequence of the current key segment, ascending or descending.
<segment_order_code></segment_order_code>	Inserts a one letter code indicating the ordering sequence of the current key segment, A or D.
<segment_position></segment_position>	Inserts the one-based position of the key segment in the record.

<segment_sequence></segment_sequence>	Inserts the ordering sequence of the current key segment, ascending or descending.
<segment_sntype></segment_sntype>	Inserts Synergy.NET data type of the field associated with the current key segment.
<segment_structure></segment_structure>	Inserts the name of the structure containing the key segment (if the segment kind is "external").
<segment_spec></segment_spec>	Inserts Synergy data type and size of the field associated with the current key segment. For example:
<segment_type></segment_type>	Inserts a word which represents the index type of the current key segment.
<segment_vbtype></segment_vbtype>	Inserts VB data type of the field associated with the current key segment.

SEGMENT_CSTYPE

Inserts C# data type of the field associated with the current key segment.

Usage

<SEGMENT_CSTYPE>

Possible values

- bool
- byte
- double
- float
- int
- long
- decimal
- DateTime
- Nullable<System.DateTime>
- sbyte
- short
- string
- uint
- ulong
- ushort

Alternate forms

None

See Also

- <SEGMENT_SNTYPE>
- <SEGMENT_SPEC>
- <SEGMENT_TYPE>
- SEGMENT_VBTYPE>

SEGMENT_DESC

Inserts the description of the field associated with the current key segment (if segment kind is field).

Usage

<SEGMENT_DESC>

Alternate forms

None

See Also

None

SEGMENT_IDXTYPE

Inserts the index type of the key segment.

Usage

<SEGMENT_IDXTYPE>

Possible values

- ALPHA
- NOCASE
- DECIMAL
- INTEGER
- UNSIGNED

Alternate forms

<segment_idxtype>

See Also

None

SEGMENT_KIND

Inserts a numeric code for the kind of the current key segment.

Usage

<SEGMENT_KIND>

Possible values

- 1 Field (see <SEGMENT_NAME> and <SEGMENT_MAPPEDNAME>)
- 2 Literal value (see <SEGMENT_LITVAL>)
- 3 External value (see <SEGMENT_STRUCTURE>)
- 4 Record number

Alternate forms

None

See Also

None

SEGMENT_LENGTH

Inserts the length of the key segment.

Usage

<SEGMENT_LENGTH>

Alternate forms

None

See Also

None

SEGMENT_LITVAL

Inserts the literal value of the key segment (if segment kind is "literal").

Usage

<SEGMENT_LITVAL>

Alternate forms

None

See Also

None

SEGMENT_MAPPEDNAME

Inserts the name of the mapped field corresponding to the current key segment (if segment kind is field).

Usage

<SEGMENT_MAPPEDNAME>

Alternate forms

- <segment_mappedname>
- <Segment_Mappedname>
- <Segment_mappedname>
- <SegmentMappedname>
- <segmentMappedname>

See Also

<SEGMENT_NAME>

SEGMENT_NAME

Inserts the name of the field associated with the current key segment (if segment kind is field).

Usage

<SEGMENT_NAME>

Alternate forms

- <segment_name>
- <Segment_Name>
- <Segment_name>
- <SegmentName>
- <segmentName>

See Also

<SEGMENT_MAPPEDNAME>

SEGMENT_NUMBER

Inserts the one-based key segment number.

Usage

<SEGMENT_NUMBER>

Possible values

• A number in the range 1 to 8.

Alternate forms

None

See Also

None

SEGMENT_ORDER

Inserts the ordering sequence of the current key segment, ascending or descending.

Usage

<SEGMENT_ORDER>

Possible values

- ASC
- DESC

Alternate forms

<segment_order>

See Also

- <SEGMENT_ORDER_CODE>
- <SEGMENT_SEQUENCE>

SEGMENT_ORDER_CODE

Inserts a one letter code indicating the ordering sequence of the current key segment, A or D.

Usage

<SEGMENT_ORDER_CODE>

Possible values

- A
- D

Alternate forms

<segment_order_code>

See Also

- <SEGMENT_ORDER>
- <SEGMENT_SEQUENCE>

SEGMENT_POSITION

Inserts the one-based position of the key segment in the record.

Usage

<SEGMENT_POSITION>

Alternate forms

None

See Also

None

SEGMENT_SEQUENCE

Inserts the ordering sequence of the current key segment, ascending or descending.

Usage

<SEGMENT_SEQUENCE>

Possible values

- ASCENDING
- DESCENDING

Alternate forms

<segment_sequence>

See Also

• <SEGMENT_ORDER>

SEGMENT_SNTYPE

Inserts Synergy.NET data type of the field associated with the current key segment.

Usage

<SEGMENT_SNTYPE>

Possible values

- byte
- DateTime
- decimal
- double
- float
- int
- long
- @Nullable<DateTime>
- short
- String
- ystem.SByte
- System.UInt16
- System.UInt32
- System.UInt64

Alternate forms

None

See Also

- <SEGMENT_CSTYPE>
- <SEGMENT_SPEC>
- <SEGMENT_TYPE>
- <SEGMENT_VBTYPE>

SEGMENT_SPEC

Inserts Synergy data type and size of the field associated with the current key segment. For example:

Usage

<SEGMENT_SPEC>

Examples of Output

- A15
- D6
- I4
- D12.2

Alternate forms

<segment_spec>

See Also

- <SEGMENT_CSTYPE>
- <SEGMENT_SNTYPE>
- <SEGMENT_TYPE>
- <SEGMENT_VBTYPE>

SEGMENT_STRUCTURE

Inserts the name of the structure containing the key segment (if the segment kind is "external").

Usage

<SEGMENT_STRUCTURE>

Alternate forms

- <segment_structure>
- <Segment_Structure>
- <Segment_structure>
- <SegmentStructure>
- <segmentStructure>

See Also

None

SEGMENT_TYPE

Inserts a word which represents the index type of the current key segment.

Usage

<SEGMENT_TYPE>

Possible values

•	ALPHA	Alpha (default)

• INTEGER Integer

• DECIMAL Decimal

• UNSIGNED Unsigned integer

• NOCASE Non case-sensitive alpha

• SEQUENCE Auto sequence (ISAM Rev 6 or higher)

• TIMESTAMP Auto timestamp, last update time (ISAM Rev 6 or higher)

• CTIMESTAMP Auto timestamp, created time (ISAM Rev 6 or higher)

NOTE: The names refer to the KEY SEGMENT TYPE not the associated FIELD DATA TYPE!

Alternate forms

<segment_type>

See Also

- <SEGMENT CSTYPE>
- <SEGMENT_SNTYPE>
- <SEGMENT_SPEC>

• <SEGMENT_VBTYPE>

SEGMENT_VBTYPE

Inserts VB data type of the field associated with the current key segment.

Usage

<SEGMENT_VBTYPE>

Possible values

- Boolean
- Byte
- DateTime
- Decimal
- Integer
- Long
- Nullable(Of System.DateTime)
- Short
- String
- System.Double
- System.SByte
- System.Single
- System.UInt16
- System.UInt32
- System.UInt64

Alternate forms

None

See Also

- SEGMENT_CSTYPE>
- <SEGMENT_SNTYPE>
- <SEGMENT_SPEC>
- <SEGMENT_TYPE>

Key Segment Loop Expression Tokens

Key segment loop expressions are expressions that can only be used within the context of a key segment loop and allow you to conditionally include or exclude code based on the attributes of the current key segment.

Key Segment Loop Expression	Description
<if seg_alpha=""></if>	The key segment is a case-sensitive alpha value.
<if seg_ascending=""></if>	The key segments data is organized in ascending sequence.
<if seg_auto_sequence=""></if>	The key segment is an auto- sequence value. ISAM Rev 6 (Synergy/DE 10.1.1 or higher).
<if seg_auto_timestamp=""></if>	The key segment is an auto-time- stamp segment, of one of the two types shown below. ISAM Rev 6 (Synergy/DE 10.1.1 or higher).
<if SEG_AUTO_TIMESTAMP_CREATED></if 	The key segment is an auto-time- stamp segment with values that are set only when records are first created. ISAM Rev 6 (Synergy/DE 10.1.1 or higher).
<if SEG_AUTO_TIMESTAMP_UPDATED></if 	The key segment is an auto-time- stamp segment with values that are set whenever records are updated. ISAM Rev 6 (Synergy/DE 10.1.1 or higher).
<if seg_decimal=""></if>	The key segment is a decimal value.
<if seg_descending=""></if>	The key segments data is organized in descending sequence.

<if seg_nocase=""></if>	The key segment is a non case- sensitive alpha value.
<if seg_signed=""></if>	The key segment is a signed integer value.
<if seg_type_external=""></if>	The key segment is associated with an external value.
<if seg_type_field=""></if>	The key segment is associated with a field in the record.
<if seg_type_literal=""></if>	The key segment is a literal value.
<if seg_type_recnum=""></if>	The key segment is a record number.
<if seg_unsigned=""></if>	The key segment is an unsigned integer value.

Key Segment Loop Expressions Reference : Associated Field Expressions

Expression	Description
<if alpha=""></if>	Data type is Alpha.
<if allow_list=""></if>	The field has allow list values.
<if alternate_name=""></if>	The field has an alternate name specified. The <field_altname> token will always return a value because if no alternate name is specified in the repository then it defaults to the fields real name. This token allows you to detect whether the alternate name is different from the real name.</field_altname>
<if array=""></if>	The field was part of an an array.

<if array_first=""></if>	The field was the very first element of an array.
<if arrive=""></if>	The field has an arrive method specified.
<if binary=""></if>	The field is a binary field. Binary fields are also alpha fields.
<if bold=""></if>	The field should be displayed in bold.
<if boolean=""></if>	The fields data type is boolean. This only applies to fields whose actual Synergy data type is boolean, not to numeric fields with a coerced type of boolean.
<if bzero=""></if>	The numeric field should be displayed blank if the field value is zero.
<if break=""></if>	The field is a break field. This expression matches any type of break condition (change, always or return).
<if break_always=""></if>	The field is a break field. The break field logic always executes whenever the field is processed.
<if break_change=""></if>	The field is a break field. The break field logic only executes when the value of the field changes.
<if break_return=""></if>	The field is a break field. The break field logic only executes when the user presses RETURN on the field.
<if change=""></if>	The field has a change method specified.
<if checkbox=""></if>	The field should be displayed as a check-box.
<if coerce_boolean=""></if>	The field data type is coerced to boolean.

<if combobox=""></if>	The field should be displayed as a combobox.
<if date=""></if>	The field is a date or nullable field. Note that for compatibility with <i>xf</i> NetLink clients, which treat julian dates (YYPP or YYYYPP) as strings, this expression does NOT consider such fields to be dates.
<if dateortime=""></if>	The field is a date field or a time field.
<if datetoday=""></if>	The date field should default to the current date.
<date_julian></date_julian>	The field is a date field of type YYJJJ or YYYYJJJ.
<date_not_julian></date_not_julian>	The field is a date field but is NOT a Julian date. It could be a reverse date or a period date.
<date_not_nullable></date_not_nullable>	The field is a date, but not a nullable date.
<date_not_period></date_not_period>	The field is a date field but is NOT a period date. It could be a reverse date or a Julian date.
<date_not_ymd></date_not_ymd>	The field is a date field but is NOT a YYMMDD or YYYYMMDD date.
<date_not_yyyymmdd< td=""><td>The field is a date field but is NOT a YYYYMMDD date.</td></date_not_yyyymmdd<>	The field is a date field but is NOT a YYYYMMDD date.
<date_nullable></date_nullable>	The field is a NULLABLE date.
<date_period></date_period>	The field is a date field of type YYPP or YYYYPP.
<date_ymd></date_ymd>	The field is a date field of type YYMMDD or YYYYMMDD.

<date_yymmdd></date_yymmdd>	The field is a date field of type YYMMDD.
-----------------------------	---

<date yyyymmdd=""></date>	The field is a date field of type
DATE_TTTTWWDD>	YYYYMMDD.

<date yyjjj=""></date>	The field is a date field of type YYJJJ.
-D1111 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	The field is a date field of type I I bbb

<DATE_YYYYJJJ> The field is a date field of type YYYYJJJ.

<DATE_YYPP> The field is a date field of type YYPP.

<DATE_YYYYPP> The field is a date field of type YYYYPP.

The field data type is decimal. Note that date <IF DECIMAL> and time fields are NOT treated as decimal

fields with this conditional test.

<IF DEFAULT> The field has a default value specified.

<IF DESCRIPTION> The field has a description specified.

<IF DISABLED>
The field is defined as being initially

disabled.

<IF DISPLAY> The field has a display method specified.

<IF DISPLAY_LENGTH> The field has a display length specified.

<IF DRILL> The field has a drill method specified.

<IF ECHO>
The field should echo characters as they are

typed.

<IF EDITFORMAT> The field has an edit format method specified.

<IF ENABLED> The field is defined as being initially enabled.

<IF FORMAT> The field has a display format specified.

ALL LINES 6	The field is an ENUM field. This is a real
<if enum=""></if>	language enumeration, not a UI Toolkit

enumerated field.

The field has a UI Toolkit enumeration

specified. This is not the same as being a real

language ENUM field.

<IF FIELD_POSITION> The field has a UI field position specified.

<IF HEADING> The field has a report heading specified.

<IF HELPID> The field has a help identifier specified.

<IF HYPERLINK> The field has a hyperlink method specified.

<IF I1> The field is an I1.

<IF ENUMERATED>

<IF I2> The field is an I2.

<IF I4> The field is an I4.

<IF I8> The field is an I8.

<IF I124> The field is an I1, I2 or I4.

<IF INCREMENT> The field is an auto-increment field.

<IF INFOLINE> The field has information line text specified.

<IF INPUT_CENTER> Input should be performed center justified.

<IF INTEGER> The field data type is defined as integer.

<IF LANGUAGE> The field is available to the language.

<if leave=""></if>	The field has a leave method specified.
<if longdesc=""></if>	The field has a long description specified.
<if mapped=""></if>	The field has a CodeGen field mapping specified.
<if negative_allowed=""></if>	The field allows negative values. Use of this expression is only appropriate for numeric fields.
<if negative_orzero=""></if>	The field allows negative values or a value of zero. Positive values are not allowed. Use of this expression is only appropriate for numeric fields.
<if negative_required=""></if>	The field requires a negative value. Use of this expression is only appropriate for numeric fields.
<if noallow_list=""></if>	The field does not have allow list values specified.
<if noalternate_name=""></if>	The field does not have an alternate name specified. The <field_altname> token will always return a value because if no alternate name is specified in the repository then it defaults to the fields real name. This token allows you to detect whether the alternate name is the same as the real name.</field_altname>
<if noarrive=""></if>	The field does not have an arrive method.
<if nobreak=""></if>	The field id not a break field.
<if nochange=""></if>	The field does not have a change method.
<if nocheckbox=""></if>	The field should not be displayed as a checkbox.

<if nocoerceboolean=""></if>	The field data type is not coerced to boolean.
<if nodefault=""></if>	The field does not have a default value.
<if nodescription=""></if>	The field does not have a description.
<if nodisplay=""></if>	The field does not have a display method.
<if nodisplay_length=""></if>	The field does not have a display length.
<if nodrill=""></if>	The field does not have a drill method.
<if noecho=""></if>	The field should not echo characters as they are typed.
<if noeditformat=""></if>	The field does not have an edit format method.
<if noformat=""></if>	The field does not have a format string.
<if nohelpid=""></if>	The field does not have a help ID.
<if nohyperlink=""></if>	The field does not have a hyperlink method.
<if noincrement=""></if>	The field is not an auto increment field.
<if noinfoline=""></if>	The field does not have information line text.
<if nolanguage=""></if>	The field is marked as "Excluded by Language".
<if noleave=""></if>	The field does not have a leave method.
<if nolongdesc=""></if>	The field does not have a long description specified.
<if nonegative=""></if>	The field does not allow negative values. This expression should only be used with numeric fields.

<if nopaintchar=""></if>	The field does not have a custom paint character.
<if noprecision=""></if>	The field does not have a decimal precision specified (i.e. it is not an implied decimal field).
<if noprompt=""></if>	The field does not have a prompt.
<if norange=""></if>	The field does not have a numeric range specified. This expression should only be used for numeric fields.
<if noreport=""></if>	The field is marked as "Excluded by ReportWriter".
<if noselections=""></if>	The field does not have selection list values.
<if noselwnd=""></if>	The field does not have a selection window name specified.
<if notalpha=""></if>	The field data type is not defines as Alpha.
<if notbinary=""></if>	The field is not a binary field. Binary fields are also alpha fields.
<if notboolean=""></if>	The field is not a boolean field.
<if notbzero=""></if>	The numeric field should not be displayed as blank if the value is zero.
<if notdate=""></if>	The field is not defined as a date field.
<if nodateortime=""></if>	The field is not a date field and not a time field.
<if notdatetoday=""></if>	The date field should not default to the current date.

<if notdecimal=""></if>	The field data type is not defined as decimal. Note that date and time fields do not count as decimal with this conditional test.
<if notenum=""></if>	The field is not a language ENUM field.
<if notenumerated=""></if>	The field is not a UI Toolkit enumerated field.
<if notimeout=""></if>	The field is not a UI Toolkit enumerated field.
<if notinteger=""></if>	The field data type is not defined as integer.
<if notnumeric=""></if>	The field data type is not a numeric type.
<if notoverlay=""></if>	The field is not an overlay field or overlay group.
<if notradiobuttons=""></if>	The field should not be viewed as a collection of radio-buttons.
<if notstructfield=""></if>	The field is not a struct field.
<if nottime=""></if>	The field is not marked as a time field.
<if notoolkit=""></if>	The field is marked as "Excluded by Toolkit".
<if notuppercase=""></if>	The field should not be converted to uppercase. Only appropriate for alpha fields.
<if notuser=""></if>	The field is not a user-defined field type.
<if nousertext=""></if>	The field does not have user text specified.
<if notusertimestamp=""></if>	The field is not a user-defined time-stamp field. (See <field_sqltype> for more information).</field_sqltype>
<if noview_length=""></if>	The field does not have a view length specified.

<if noweb=""></if>	The field is marked as "Excluded by Web".
<if numeric=""></if>	The field data type is defines as a numeric type.
<if ocnative=""></if>	The fields Objective-C data type is a native type (e.g. int or BOOL).
<if ocobject=""></if>	The fields Objective-C data type is an object type (e.g. NSString or NSDate).
<if optional=""></if>	The field is marked as optional (i.e. not required).
<if overlay=""></if>	The field is an overlay field or overlay group.
<if painchar=""></if>	The field has a custom paint character specified.
<if precision=""></if>	The decimal field has a precision specified (i.e. it is an implied decimal field).
<if prompt=""></if>	The field has a prompt specified.
<if prompt_position=""></if>	The field has a UI prompt position specified.
<if radiobuttons=""></if>	The field should be viewed as a collection of radio-buttons.
<if range=""></if>	The field has a numeric range (minimum/maximum values) specified.
<if readonly=""></if>	The field state is specified as initially read- only.
<if readwrite=""></if>	The field state is specified as initially writable.
<if report=""></if>	The field is available to ReportWriter.

<if report_center=""></if>	The field should be printed center justified on reports.
<if report_left=""></if>	The field should be printed left justified on reports.
<if report_right=""></if>	The field should be printed right justified on reports.
<if required=""></if>	The field is marked as required (i.e. must contain a valid value).
<if reverse=""></if>	The field should be displayed in reverse video.
<if selections=""></if>	The field has one or more selection list values.
<if selwnd=""></if>	The field has a selection window name specified.
<if structfield=""></if>	The field is a struct field.
<if textbox=""></if>	The field should be displayed as a text-box.
<if time=""></if>	The decimal fields' type is specified as a time field.
<if time_hhmm=""></if>	The field is a time field of type HHMM.
<if time_hhmmss=""></if>	The field is a time field of type HHMMSS.
<if timeout=""></if>	The field has an input timeout specified.
<if timenow=""></if>	The time field should default to the current time.
<if toolkit=""></if>	The field is available to UI Toolkit.
	The field content should be displayed

<if underline=""> u</if>	nderlined.
--------------------------	------------

<pre></pre>	ys be converted to	
-------------	--------------------	--

uppercase.

<IF USER> The field is a user defines field type.

<IF USERTEXT> The field has user text specified.

The field is a user-defined time-stamp field.

information).

<IF VIEW_LENGTH> The field has a view length specified.

<IF WEB> The field is available to the web products.

Relation Loops

A relation loop is a template file construct which allows you to iterate through the collection of relations that CodeGen has information about. In order to use a relation loop you must be processing a repository structure, either directly via the —s command line option, or because the structure is referenced by the UI Toolkit input window that you are processing via the —w command line option.

Relation loops are delimited by a matching pair of <**RELATION_LOOP**> and </**RELATION_LOOP**> tags that surround the template code to be inserted for each relation.

The code between the opening and closing tags of a relation loop is repeated for each relation in the structure being processed. This code within a relation loop can contain other generic and structure tokens, and can also contain special relation loop tokens, which are discussed later. Relation loop tokens can only be used inside a relation loop, and a relation loop can't be declared within most other loop constructs (except structure loops).

Relation loop tokens are tokens that can only be used within a relation loop:

- Relation Loop Expansion Tokens
- Relation Loop Expression Tokens

There are two types of relation loop, multi-line relation loops and in-line relation loops.

Multi-Line Relation Loops

Multi-line relation loops occur when the opening and closing relation loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
[code]

<RELATION_LOOP>code

</RELATION LOOP>
```

[code]

Multi-line relation loops generate one or more lines of output code for each relation that is processed.

In-Line Relation Loops

In-line relation loops exist when the opening and closing relation loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

[code] <RELATION_LOOP> code </RELATION_LOOP> [code]

In-line relation loops generate code into the current output line only.

Relation Loop Expansion Tokens

Relation loop expansion tokens are used to insert information about the current relation being processed within the context of a relation loop into the output stream. Relation loop tokens can only be used inside a relation loop.

Key Loop Token	Description
<relation_number></relation_number>	Inserts the one-based index of the current relation.
<relation_name></relation_name>	Inserts the name of the current relation.
<relation_fromkey></relation_fromkey>	Inserts the name of the key in the current structure from which the relation originates.
<relation_tostructure></relation_tostructure>	Inserts the name of the structure targeted by this relation.
<relation_tostructure_noplural></relation_tostructure_noplural>	Inserts the name of the structure targeted by this relation. If the name of the structure ends with an "S" character it will be removed.
<relation_tostructure_plural></relation_tostructure_plural>	Inserts the name of the structure targeted by this relation. If the name of the structure does not end with an "S" character one will be added.
<relation_tokey></relation_tokey>	Inserts the name of the key in the structure targeted by this relation.
<from_key_changes></from_key_changes>	Inserts text indicating whether the from key allows changes to existing key entries.
<from_key_density></from_key_density>	Inserts the key packing density for the from key.

<FROM KEY DESCRIPTION> <FROM KEY DUPLICATES> <FROM_KEY_DUPLICATES_AT> <FROM_KEY_LENGTH> <FROM KEY NULLTYPE> <FROM KEY NULLVALUE> <FROM KEY NUMBER> <FROM KEY ORDER> <FROM KEY SEGMENTS> <FROM_KEY_UNIQUE>

Inserts the description of the from key. Be aware that key descriptions are optional in Repository, so this token may expand to nothing! Inserts text indicating whether the from key allows duplicate values. Inserts text indicating where duplicate values are inserted for the from key. Inserts the length of the from key. The length of the key is the sum of the length of all key segments. Inserts the from keys null key type. The token should only be used within a <IF NULLKEY> expression. Inserts the from keys null key value. The token should only be used within an <IF NULLKEY> expression. Inserts the number of the from key, primary key is key 0, first alternate key is key 1, and so on. Inserts the ordering sequence of the from key, ascending or descending. Inserts the number of from key segments defined. If the from key does not allow duplicates then inserts the text UNIQUE, otherwise

nothing is inserted.

Inserts text indicating

<to_key_changes></to_key_changes>	whether the to key allows changes to existing key entries.
<to_key_density></to_key_density>	Inserts the key packing density for the to key.
<to_key_description></to_key_description>	Inserts the description of the to key. Be aware that key descriptions are optional in Repository, so this token may expand to nothing!
<to_key_duplicates></to_key_duplicates>	Inserts text indicating whether the to key allows duplicate values.
<to_key_duplicates_at></to_key_duplicates_at>	Inserts text indicating where duplicate values are inserted for the to key.
<to_key_length></to_key_length>	Inserts the length of the to key. The length of the key is the sum of the length of all key segments.
<to_key_nulltype></to_key_nulltype>	Inserts the to keys null key type. The token should only be used within a <if nullkey=""> expression.</if>
<to_key_nullvalue></to_key_nullvalue>	Inserts the to keys null key value. The token should only be used within an <if nullkey=""> expression.</if>
<to_key_number></to_key_number>	Inserts the number of the to key, primary key is key 0, first alternate key is key 1, and so on.
<to_key_order></to_key_order>	Inserts the ordering sequence of the to key, ascending or descending.
<to_key_segments></to_key_segments>	Inserts the number of to key segments defined.



If the to key does not allow duplicates then inserts the text UNIQUE, otherwise nothing is inserted.

RELATION_NUMBER

Inserts the one-based index of the current relation

Usage

<RELATION_NUMBER>

Alternate forms

None

See Also

None

RELATION_NAME

Inserts the name of the current relation.

Usage

<RELATION_NAME>

Alternate forms

None

See Also

None

RELATION_FROMKEY

Inserts the name of the key in the current structure from which the relation originates.

Usage

<RELATION_FROMKEY>

Alternate forms

- <relation_fromkey>
- <Relation_Fromkey>
- <Relation_fromkey>
- <RelationFromkey>
- <relationFromkey>

See Also

None

RELATION_TOSTRUCTURE

Inserts the name of the structure targeted by this relation.

Usage

<RELATION_TOSTRUCTURE>

Alternate forms

- <relation_tostructure>
- <Relation_Tostructure>
- <Relation_tostructure>
- <RelationTostructure>
- <relationTostructure>

See Also

None

RELATION_TOSTRUCTURE_NOPLURAL

Inserts the name of the structure targeted by this relation. If the name of the structure ends with an "S" character it will be removed.

Usage

<RELATION_TOSTRUCTURE_NOPLURAL>

Alternate forms

- <relation_tostructure_noplural>
- <Relation_Tostructure_Noplural>
- <Relation_tostructure_noplural>
- <RelationTostructureNoplural>
- <relationTostructureNoplural>

See Also

None

REPLATION_TOSTRUCTURE_PLURAL

Inserts the name of the structure targeted by this relation. If the name of the structure does not end with an "S" character one will be added.

Usage

<RELATION_TOSTRUCTURE_PLURAL>

Alternate forms

- <relation_tostructure_plural>
- <Relation_Tostructure_Plural>
- <Relation_tostructure_plural>
- <RelationTostructurePlural>
- <relationTostructurePlural>

See Also

None

RELATION_TOKEY

Inserts the name of the key in the structure targeted by this relation.

Usage

<RELATION_TOKEY>

Alternate forms

- <relation_tokey>
- <Relation_Tokey>
- <Relation_tokey>
- <RelationTokey>
- <relationTokey>

See Also

None

FROM_KEY_CHANGES

Inserts text indicating whether the FROM key allows changes to existing key entries.

Usage

<FROM_KEY_CHANGES>

Possible values

- CHANGES
- NOCHANGES

Alternate forms

None

See Also

None

FROM_KEY_DENSITY

Inserts the key packing density for the FROM key.

Usage

<FROM_KEY_DENSITY>

Possible values

• A number in the range 50 to 100.

Alternate forms

None

See Also

None

FROM_KEY_DESCRIPTION

Inserts the description of the FROM key. Be aware that key descriptions are optional in Repository, so this token may expand to nothing.

Usage

<FROM_KEY_DESCRIPTION>

Alternate forms

None

See Also

None

FROM_KEY_DUPLICATES

Inserts text indicating whether the FROM key allows duplicate values.

Usage

<FROM_KEY_DUPLICATES>

Possible values

- DUPLICATES
- UNIQUE

Alternate forms

None

See Also

None

FROM_KEY_DUPLICATES_AT

Inserts text indicating where duplicate values are inserted for the FROM key.

Usage

<FROM_KEY_DUPLICATES_AT>

Possible values

- FRONT
- END

Alternate forms

None

See Also

None

FROM_KEY_LENGTH

Inserts the length of the FROM key. The length of the key is the sum of the length of all key segments.

Usage

<FROM_KEY_LENGTH>

Alternate forms

None

See Also

None

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FROM_KEY_NULLTYPE

Inserts the FROM keys null key type. The token should only be used within a <IF NULLKEY> expression.

Usage

<FROM_KEY_NULLTYPE>

Possible values

- NONE
- REPLICATE
- NOREPLICATE
- SHORT

Alternate forms

<key_nulltype>

See Also

None

FROM_KEY_NULLVALUE

Inserts the FROM keys null key value. The token should only be used within an <IF NULLKEY> expression.

Usage

<FROM_KEY_NULLVALUE>

Alternate forms

None

See Also

None

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FROM_KEY_NUMBER

Inserts the number of the FROM key, primary key is key 0, first alternate key is key 1, and so on.

Usage

<FROM_KEY_NUMBER>

Possible values

• A number in the range 0 to 1023.

Alternate forms

None

See Also

None

FROM_KEY_ORDER

Inserts the ordering sequence of the FROM key, ascending or descending.

Usage

<FROM_KEY_ORDER>

Possible values

- ASC
- DESC

Alternate forms

None

See Also

None

FROM_KEY_SEGMENTS

Inserts the number of FROM key segments defined.

Usage

<FROM_KEY_SEGMENTS>

Possible values

• A number in the range 1 to 8.

Alternate forms

None

See Also

None

FROM_KEY_UNIQUE

If the FROM key does not allow duplicates then inserts the text UNIQUE, otherwise nothing is inserted.

Usage

<FROM_KEY_UNIQUE>

Alternate forms

None

See Also

None

TO_KEY_CHANGES

Inserts text indicating whether the TO key allows changes to existing key entries.

Usage

<TO_KEY_CHANGES>

Possible values

- CHANGES
- NOCHANGES

Alternate forms

None

See Also

None

TO_KEY_DENSITY

Inserts the key packing density for the TO key.

Usage

<TO_KEY_DENSITY>

Possible values

• A number in the range 50 to 100.

Alternate forms

None

See Also

None

TO_KEY_DESCRIPTION

Inserts the description of the TO key. Be aware that key descriptions are optional in Repository, so this token may expand to nothing!

Usage

<TO_KEY_DESCRIPTION>

Alternate forms

None

See Also

None

TO_KEY_DUPLICATES

Inserts text indicating whether the TO key allows duplicate values.

Usage

<TO_KEY_DUPLICATES>

Possible values

- DUPLICATES
- UNIQUE

Alternate forms

None

See Also

None

TO_KEY_DUPLICATES_AT

Inserts text indicating where duplicate values are inserted for the TO key.

Usage

<TO_KEY_DUPLICATES_AT>

Possible values

- FRONT
- END

Alternate forms

None

See Also

None

TO_KEY_LENGTH

Inserts the length of the TO key. The length of the key is the sum of the length of all key segments.

Usage

<TO_KEY_LENGTH>

Alternate forms

None

See Also

None

TO_KEY_NULLTYPE

Inserts the TO keys null key type. The token should only be used within a <IF NULLKEY> expression.

Usage

<TO_KEY_NULLTYPE>

Possible values

- NONE
- REPLICATE
- NOREPLICATE
- SHORT

Alternate forms

<key_nulltype>

See Also

None

TO_KEY_NULLVALUE

Inserts the TO keys null key value. The token should only be used within an <IF NULLKEY> expression.

Usage

<TO_KEY_NULLVALUE>

Alternate forms

None

See Also

None

Copyright @ 2018 Synergex International, Inc.

TO_KEY_NUMBER

Inserts the number of the TO key, primary key is key 0, first alternate key is key 1, and so on.

Usage

<TO_KEY_NUMBER>

Possible values

• A number in the range 0 to 1023.

Alternate forms

None

See Also

None

TO_KEY_ORDER

Inserts the ordering sequence of the TO key, ascending or descending.

Usage

<TO_KEY_ORDER>

Possible values

- ASC
- DESC

Alternate forms

None

See Also

None

TO_KEY_SEGMENTS

Inserts the number of TO key segments defined.

Usage

<TO_KEY_SEGMENTS>

Possible values

• A number in the range 1 to 8.

Alternate forms

None

See Also

None

TO_KEY_UNIQUE

If the TO key does not allow duplicates then inserts the text UNIQUE, otherwise nothing is inserted.

Usage

<TO_KEY_UNIQUE>

Alternate forms

None

See Also

None

Relation Loop Expression Tokens

Relation loop expressions are expressions that can only be used within the context of a relation loop and allow you to conditionally include or exclude code based on the attributes of the current key relation.

Relation Loop Expression	Description
<if from_access=""></if>	The relations "from" key is an access key.
<if from_ascending=""></if>	The relations "from" key order is ascending.
<if from_auto_sequence=""></if>	The relations "from" key is an auto-sequence key.
<if from_auto_timestamp=""></if>	The relations "from" key is an auto-timestamp key.
<if FROM_AUTO_TIMESTAMP_CREATED></if 	The relations "from" key is an auto-timestamp key (time record created).
<if from_auto_timestamp_updated=""></if>	The relations "from" key is an auto-timestamp key (time record last updated).
<if from_changes=""></if>	The relations "from" key allows key value changes.
<if from_descending=""></if>	The relations "from" key order is descending.
<if from_duplicates=""></if>	The relations "from" key allows duplicate key values.
<if from_duplicatesatend=""></if>	The relations "from" key inserts duplicate key values after existing duplicates (FIFO).
<if from_duplicatesatfront=""></if>	The relations "from" key inserts duplicate key values before existing duplicates (LIFO).
<if from_first_seg_nocase=""></if>	The relations "from" key's first segment type is non-case-sensitive alpha.

<if from_foreign=""></if>	The relations "from" key is a foreign key.
<pre><if from_multiple_segments=""></if></pre>	The relations "from" key has multiple segments.
<if from_nochanges=""></if>	The relations "from" key does not allow key value changes.
<if from_noduplicates=""></if>	The relations "from" key does not allow duplicate key values.
<if from_nullkey=""></if>	The relations "from" key is a null-replicating key.
<if from_nullvalue=""></if>	The relations "from" key has a null-replicating key value.
<if from_single_segment=""></if>	The relations "from" key has a single segment.
<if one_way_one_to_many=""></if>	The relationship is a one-to- many relationship and a matching relationship is not defined in the opposite direction.
<if one_way_one_to_one=""></if>	The relationship is a one-to-one relationship and a matching relationship is not defined in the opposite direction.
<if to_ascending=""></if>	The relations "to" key order is ascending.
<if to_auto_sequence=""></if>	The relations "to" key is an autosequence key.
<if to_auto_timestamp=""></if>	The relations "to" key is an auto-timestamp key.
<if TO_AUTO_TIMESTAMP_CREATED></if 	The relations "to" key is an auto- timestamp key (time record created).
<if TO_AUTO_TIMESTAMP_UPDATED></if 	The relations "to" key is an auto- timestamp key (time record last updated).
<if to_changes=""></if>	The relations "to" key allows key value changes.

<if to_descending=""></if>	The relations "to" key order is descending.
<if to_duplicates=""></if>	The relations "to" key allows duplicate key values.
<if to_duplicatesatend=""></if>	The relations "to" key inserts duplicate key values after existing duplicates (FIFO).
<if to_duplicatesatfront=""></if>	The relations "to" key inserts duplicate key values before existing duplicates (LIFO).
<if to_first_seg_nocase=""></if>	The relations "to" key's first segment type is non-case-sensitive alpha.
<if to_longer=""></if>	The relations "to" key is longer than the "from" key.
<pre><if to_multiple_segments=""></if></pre>	The relations "to" key has multiple segments.
<if to_nochanges=""></if>	The relations "to" key does not allow key value changes.
<if to_noduplicates=""></if>	The relations "to" key does not allow duplicate key values.
<if to_nullkey=""></if>	The relations "to" key is a null-replicating key.
<if to_nullvalue=""></if>	The relations "to" key has a null key value specified.
<if to_same_size=""></if>	The relations "to" key is the same size as the "from" key.
<if to_shorter=""></if>	The relations "to" key is shorter than the "from" key.
<if to_single_segment=""></if>	The relations "to" key has a single segment. The relationship is a one-to-
<if two_way_one_to_many=""></if>	many relationship and a matching relationship is also defined in the opposite direction.

The relationship is a one-to-one

<if th="" two_v<=""><th>NAY_</th><th>ONE_</th><th>_TO_</th><th>ONE></th></if>	NAY_	ONE_	_TO_	ONE>
--	------	------	------	------

relationship and a matching relationship is also defined in the opposite direction.

Relation Loop Key Segment Loops

There are two special loop constructs that may be used within a relation loop. They are the "from key segment loop" and the "to key segment loop". These loops allow you to iterate through the key segments associated with the current relations "from key" and "to key".

Both of these loops behave just like a key segment loop within a key loop. All key segment loop expansion tokens and key segment loop expression tokens can be used in both of these loops. In addition for key segments associated with structure fields, you can also use field loop expansion tokens and field loop expression tokens.

From Key Segment Loops

From key segment loops allow you to iterate through the key segments associated with the key in the current structure that is identified as the relations "from key". From key segment loops may be in a multi-line form:

```
<RELATION_LOOP>
[code]
<FROM_KEY_SEGMENT_LOOP>
code</FROM_KEY_SEGMENT_LOOP>
[code]
</RELATION_LOOP>
```

Or they may be in an in-line form:

```
<RELATION_LOOP>
[code]
[code] <FROM_KEY_SEGMENT_LOOP> code
</FROM_KEY_SEGMENT_LOOP> [code]
[code]
</RELATION_LOOP>
```

To Key Segment Loops

To key segment loops allows you to iterate through the key segments

associated with the key in a different structure that is identified as the relations "to key". To key segment loops may be in a multi-line form:

```
<RELATION_LOOP>
[code]
<TO_KEY_SEGMENT_LOOP>
code
</TO_KEY_SEGMENT_LOOP>
[code]
</RELATION_LOOP>
```

Or they may be in an in-line form:

```
<RELATION_LOOP>
[code]
[code] <TO_KEY_SEGMENT_LOOP> code
</TO_KEY_SEGMENT_LOOP> [code]
[code]
</RELATION_LOOP>
```

Enumeration Loops

An enumeration loop is a template file construct which allows you to iterate through the collection of enumerations that CodeGen has information about. In order to use an enumeration loop your repository must include at least one enumeration definition.

Enumeration loops are delimited by a matching pair of **<ENUM_LOOP>** and **</ENUM_LOOP>** tags that surround the template code to be inserted for each enumeration.

Enumeration loop tokens are tokens that can only be used within an enumeration loop:

- Enumeration Loop Expansion Tokens
- Enumeration Loop Expression Tokens

There are three types of enumeration loop, multi-line enumeration loops, inline enumeration loops, and structure enumeration loops.

Multi-line Enumeration Loops

Multi-line enumeration loops occur when the opening and closing key loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
[code]

ENUM_LOOP>code

(ENUM_LOOP>

[code]
```

Multi-line enumeration loops generate one or more lines of output code for each enumeration that is processed.

In-Line Enumeration Loops

In-line enumeration loops exist when the opening and closing enumeration

loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
[code] <ENUM_LOOP> code </ENUM_LOOP> [code]
```

In-line enumeration loops generate code into the current output line only.

The code between the opening and closing tags of an enumeration loop is repeated for each enumeration in the repository.

The code within an enumeration loop can contain generic tokens, and can also contain special enumeration loop tokens, which are discussed later.

Enumeration loop tokens can only be used inside an enumeration loop, and an enumeration loop can't be declared within any other loop construct.

Structure Enumeration Loops

A structure enumeration loop is similar to an enumeration loop, except that only enumerations that are referenced by fields in the repository structure that is being processed are included. As with enumeration loops, CodeGen supports both multi-line and in-line structure enumeration loops.

A multi-line structure enumeration loop looks like this:

```
[code]

<ENUM_LOOP_STRUCTURE>
code
</ENUM_LOOP_STRUCTURE>
[code]
```

And an in-line structure enumeration loop looks like this:

```
[code] <ENUM_LOOP_STRUCTURE> code </ENUM_LOOP_STRUCTURE> [code]
```

Multi-line structure enumeration loops generate one or more lines of output code for each enumeration that is processed, whereas in-line structure enumeration loops generate code into the current output line only.

Enumeration Loop Expansion Tokens

Enumeration loop expansion tokens are used to insert information about the current enumeration being processed within the context of an enumeration loop into the output stream. Enumeration loop tokens can only be used inside an enumeration loop.

Enumeration Loop Token	Description
<enum_count></enum_count>	Inserts the total number of repository enumerations being processed.
<enum_description></enum_description>	Inserts the short description of the enumeration.
<enum_long_description></enum_long_description>	Inserts the long description of the enumeration.
<enum_member_count></enum_member_count>	Inserts the number of members that comprise to the enumeration.
<enum_name></enum_name>	Inserts the name of the enumeration.
<enum_number></enum_number>	Inserts the one-based index within the collection of enumerations being processed, of the enumeration that is currently being processed.

ENUM_COUNT

Inserts the total number of repository enumerations being processed. For regular enumeration loops this will be the number of enumerations defined in the repository. For structure enumeration loops this will be the number of enumerations referenced by the repository structure being processed.

Usage

<ENUM_COUNT>

Alternate forms

None

See Also

<ENUM_NUMBER>

ENUM_DESCRIPTION

Inserts the short description of the enumeration.

You can test whether an enumeration has a description using the enumeration loop <IF DESCRIPTION> expression.

Usage

<ENUM_DESCRIPTION>

Alternate forms

None

See Also

None

ENUM_LONG_DESCRIPTION

Inserts the long description of the enumeration.

You can test whether an enumeration has a long description using the enumeration loop <IF LONG_DESCRIPTION> expression.

Usage

<ENUM_LONG_DESCRIPTION>

Alternate forms

None

See Also

None

ENUM_MEMBER_COUNT

Inserts the number of members that comprise to the enumeration.

Usage

<ENUM_MEMBER_COUNT>

Alternate forms

None

See Also

None

ENUM_NAME

Inserts the name of the enumeration.

Usage

<ENUM_NAME>

Alternate forms

- <enum_name>
- <Enum_Name>
- <Enum_name>
- <EnumName>
- <enumName>

See Also

None

ENUM_NUMBER

Inserts the one-based index within the collection of enumerations being processed, of the enumeration that is currently being processed.

Usage

<ENUM_NUMBER>

Alternate forms

None

See Also

<ENUM_COUNT>

Enumeration Loop Expression Tokens

Enumeration loop expressions are expressions that can only be used within the context of an enumeration loop and allow you to conditionally include or exclude code based on the attributes of the current enumeration.

Enumeration Loop Expression	Description
<if description=""></if>	The enumeration has descriptive text specified.
<if long_description=""></if>	The enumeration has a long description specified.

Enumeration Member Loops

An enumeration member loop is a template file construct which allows you to iterate through the collection of members for a particular enumeration that CodeGen has information about. In order to use an enumeration member loop your repository must include the definition of at least one enumeration.

Enumeration member loops are delimited by a matching pair of <ENUM_MEMBER_LOOP> and </ENUM_MEMBER_LOOP> tags, which surround the template code to be inserted for each enumeration member.

The code between the opening and closing tags of an enumeration member loop is repeated for each enumeration member in the enumeration being processed. This code within an enumeration member loop can contain generic tokens, and can also contain special enumeration member loop tokens, which are discussed later. Enumeration member loop tokens can only be used inside an enumeration member loop, and an enumeration member loop can only be declared within one of the supported enumeration loop constructs.

Enumeration member loop tokens are tokens that can only be used within an enumeration member loop:

- Enumeration Member Loop Expansion Tokens
- Enumeration Member Loop Expression Tokens

There are two types of enumeration member loop, multi-line enumeration member loops, and on-line enumeration member loops.

Multi-Line Enumeration Member Loops

Multi-line enumeration member loops occur when the opening and closing enumeration member loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
<ENUM_LOOP>[code]
<ENUM_MEMBER_LOOP>
code
</ENUM_MEMBER_LOOP>
```

```
[code] </ENUM LOOP>
```

Multi-line enumeration member loops generate one or more lines of output code for each enumeration member that is processed.

In-Line Enumeration Member Loops

In-line enumeration member loops exist when the opening and closing enumeration member loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
<ENUM_LOOP>
[code]
[code] <ENUM_MEMBER_LOOP> code
</ENUM_MEMBER_LOOP> [code]
[code]
</ENUM_LOOP>
```

In-line enumeration member loops generate code into the current output line only.

Enumeration Member Loop Expansion Tokens

Enumeration member loop expansion tokens are used to insert information about the current enumeration member being processed within the context of an enumeration member loop into the output stream. Enumeration member loop tokens can only be used inside an enumeration member loop.

Enumeration Member Loop Token	Description
<enum_member_name></enum_member_name>	Inserts the name of the enumeration member.
<enum_member_explicit_value></enum_member_explicit_value>	Inserts the numeric value of the enumeration member that is explicitly specified in the repository.
<enum_member_implicit_value></enum_member_implicit_value>	Inserts the numeric value of the enumeration member that is either explicitly specified in the repository, or is implicitly derived by following the same rules observed by the Synergy compiler.

ENUM_MEMBER_NAME

Inserts the name of the enumeration member.

Usage

<ENUM_MEMBER_NAME>

Alternate forms

- <enum_member_name>
- <Enum_Member_Name>
- <Enum_member_name>
- <EnumMemberName>
- <enumMemberName>

See Also

None

ENUM_MEMBER_EXPLICIT_VALUE

Inserts the numeric value of the enumeration member that is explicitly specified in the repository. Note that some enumeration members may not have an explicit value specified. In Synergy, enumeration members that do not have an explicit specified value will default to the value of the previous member incremented by one, except for the first member whose value defaults to zero. If a member does not have an explicitly specified value then inserts nothing.

The token is primarily used when defining enumerations in Synergy language.

Usage

<ENUM_MEMBER_EXPLICIT_VALUE>

Alternate forms

None

See Also

<ENUM_MEMBER_IMPLICIT_VALUE>

ENUM_MEMBER_IMPLICIT_VALUE

Inserts the numeric value of the enumeration member that is either explicitly specified in the repository, or is implicitly derived by following the same rules observed by the Synergy compiler. This token will always cause a value to be inserted, even if the enumeration member being processed does not have an explicitly defined value.

This token is primarily used when defining enumerations in languages other than Synergy language.

Usage

<ENUM_MEMBER_IMPLICIT_VALUE>

Alternate forms

None

See Also

<ENUM_MEMBER_EXPLICIT_VALUE>

Enumeration Member Loop Expression Tokens

Enumeration member loop expressions are expressions that can only be used within the context of an enumeration member loop and allow you to conditionally include or exclude code based on the attributes of the current enumeration member.

Enumeration Member Loop Expression	Description
<if EXPLICIT_VALUE></if 	The enumeration member has a value explicitly defined.
<if NOEXPLICIT_VALUE></if 	The enumeration member does not have a value explicitly defined, default rules should be used to imply the members value.

Button Loops

A button loop is a template file construct which allows you to iterate through the collection of buttons that CodeGen has information about.

The definition of the buttons processed in a button loop can come from one of two places. If you are processing based on a UI Toolkit input window definition then by default the collection of buttons is determined by the buttons found in that input window definition. Otherwise the button collection is defined by a configuration file called DefaultButtons.xml, which you will find in the folder that you installed CodeGen into.

Note: Even if you are processing based on a UI Toolkit input window definition, you can chose to ignore the definition of buttons found in the window and use the definitions from DefaultButtons.xml via the –b command line option.

Button loops are delimited by a matching pair of **BUTTON_LOOP**> and **</BUTTON_LOOP**> tags that surround the template code to be inserted for each button. The code between the opening and closing tags of a button loop is repeated for each button in the button collection.

Button loop tokens are tokens that can only be used within a button loop:

- Button Loop Expansion Tokens
- Button Loop Expression Tokens

There are two types of button loop, multi-line button loops and in-line button loops.

Multi-Line Button Loops

Multi-line button loops occur when the opening and closing button loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

[code]<BUTTON_LOOP>
code

```
</BUTTON_LOOP>
[code]
```

Multi-line button loops generate one or more lines of output code for each button that is processed.

In-Line Button Loops

In-line button loops exist when the opening and closing button loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
[code] <BUTTON_LOOP> code </BUTTON_LOOP> [code]
```

In-line button loops generate code into the current output line only.

Default Buttons

The default buttons are defined in DefaultButtons.xml, as follows:

Special Button Processing

When generating application source code, and in particular forms, it is often useful to be able to identify a button as either an "OK" button (which generally saves the data and closes the form) or a "Cancel" button (which closes the form without saving the data). Unfortunately there is no way to easily identify this from information in a UI Toolkit input window script.

CodeGen considers a button to be an "OK" button if the term "OK" appears in either the name or caption of the button.

CodeGen considers a button to be a "Cancel" button if the word "Cancel" appears in the buttons name or caption, or if the word "abandon" appears in the buttons name (UI Toolkit programmers often name the cancel button O_ABANDON).

The status a button as either an OK or Cancel button can be tested with the button loop expressions <IF OKBUTTON> and <IF CANCELBUTTON>.

Button Loop Expansion Tokens

Button loop expansion tokens are used to insert information about the current button being processed within the context of a button loop into the output stream. Button loop tokens can only be used inside a button loop.

Button Loop Token	Description
<button_caption></button_caption>	Inserts the buttons caption.
<button_colpx></button_colpx>	Inserts the calculated column position for the button in pixels.
<button_elb></button_elb>	Inserts the name of the ELB containing the buttons method.
<button_image></button_image>	Inserts the name of the buttons image file.
<button_method></button_method>	Inserts the name of the buttons method.
<button_name></button_name>	Inserts the Name of the button. All buttons will have a name defined.
<button_number></button_number>	Inserts the one-based index number of the button in the windows Buttons collection.
<button_quickselect< td=""><td>Inserts the buttons explicitly specified quick-select character.</td></button_quickselect<>	Inserts the buttons explicitly specified quick-select character.
<button_rowpx></button_rowpx>	Inserts the calculated row position for the button in pixels.
<button_widthpx></button_widthpx>	Inserts the width of the button in pixels.

BUTTON_CAPTION

Inserts the buttons caption.

In UI Toolkit a button can have a caption, or an image, but not both. The <IF CAPTION> and <IF IMAGE> expressions will allow you to provide the appropriate code.

Usage

<BUTTON_CAPTION>

Alternate forms

None

See also

None

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

BUTTON_COLPX

Inserts the calculated column position for the button in pixels. The column position is calculated by adding the sizes of all preceding buttons, with 15 pixels of space allowed before each button.

Usage

<BUTTON_COLPX>

Alternate forms

None

See also

None

BUTTON_ELB

Inserts the name of the ELB containing the buttons method.

Refer to the discussion relating to <BUTTON_METHOD>. Methods are very rare, and specifying an external ELB is even rarer. You can use the expression <IF ELB> to determine if a button method ELB is specified.

<Usage>

<BUTTON_ELB>

Alternate forms

None

See also

None

BUTTON_IMAGE

Inserts the name of the buttons image file.

In UI Toolkit a button can have a caption, or an image, but not both. The <IF CAPTION> and <IF IMAGE> expressions will allow you to provide the appropriate code.

Usage

<BUTTON_IMAGE>

Alternate forms

None

See also

None

BUTTON_METHOD

Inserts the name of the buttons method.

In UI Toolkit a button method may be specified as the event handler to be executed when the button is clicked. This is rare however, most applications relying on the fact that if a button method is not specified then the name of the button is signaled as a menu entry.

Most buttons will not have a method specified. You can use the <IF METHOD> expression to determine if a method is specified.

Usage

<BUTTON_METHOD>

Alternate forms

None

See also

None

BUTTON_NAME

Inserts the Name of the button. All buttons will have a name defined.

Usage

<BUTTON_NAME>

Alternate forms

None

See also

None

BUTTON_NUMBER

Inserts the one-based index number of the button in the windows Buttons collection.

Usage

<BUTTON_NUMBER>

Alternate forms

None

See also

None

BUTTON_QUICKSELECT

Inserts the buttons explicitly specified quick-select character.

Usage

<BUTTON_QUICKSELECT>

Alternate forms

None

See also

None

BUTTON_ROWPX

Inserts the calculated row position for the button in pixels. The calculation takes into account the pixel height of the window, as well as the additional space which will have been allowed in the window height when the window size was calculated. If code generation is based on an existing UI Toolkit window then the height of the source window is used to calculate the value, otherwise the number of fields in the structure is used.

Usage

<BUTTON_ROWPX>

Alternate forms

None

See also

None

BUTTON_WIDTHPX

Inserts the width of the button in pixels. The width is calculated by multiplying the number of characters in the buttons caption by 12 pixels per character, and this value can be overridden using the -cw option. If the calculated button width is less than 75 pixels then 75 pixels is used.

Usage

<BUTTON_WIDTHPX>

Alternate forms

None

See also

None

Button Loop Expression Tokens

Button loop expressions are expressions that can only be used within the context of a button loop and allow you to conditionally include or exclude code based on the attributes of the current button.

Button Loop Expression	Description
<if CANCELBUTTON></if 	The button is a Cancel button.
<if caption=""></if>	The button has a caption specified.
<if elb=""></if>	The event handler method is specified to be in an external ELB.
<if GENERICBUTTON></if 	The button is a generic button (i.e. it is not an OK or Cancel button).
<if image=""></if>	The button has an image file specified.
<if method=""></if>	The image has an event handler method specified.
<if okbutton=""></if>	The button is an OK button.
<if QUICKSELECT></if 	The button has a quick-select character specified.

File Loops

A file loop is a template file construct which allows you to iterate through the collection of files that are assigned to the structure which is currently being processed.

File loops are delimited by a matching pair of **FILE_LOOP**> and **</FILE_LOOP**> tags that surround the template code to be inserted for each file. The code between the opening and closing tags of a file loop is repeated for each file assigned to the structure being processed.

File loop tokens are tokens that can only be used within a file loop:

- File Loop Expansion Tokens
- File Loop Expression Tokens

There are two types of file loops, called multi-line file loops, and in-line file loops.

Multi-Line File Loops

Multi-line file loops occur when the opening and closing file loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
[code]
<FILE_LOOP>code
</FILE_LOOP>
[code]
```

Multi-line file loops generate one or more lines of output code for each file that is processed.

In-Line File Loops

In-line file loops exist when the opening and closing file loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

[code] **<FILE_LOOP>** code **</FILE_LOOP>** [code]

In-line file loops generate code into the current output line.

File Loop Expansion Tokens

File loop expansion tokens are used to insert information about the current file being processed within the context of a file loop into the output stream. File loop tokens can only be used inside a file loop.

File Loop Token	Description
<floop_addressing></floop_addressing>	Inserts the addressing size of the current data file.
<pre><floop_change_tracking< pre=""></floop_change_tracking<></pre>	Inserts a string indicating whether the structures first assigned file has change tracking enabled.
<floop_compression></floop_compression>	Inserts a string indicating whether the current file has data compression enabled.
<floop_desc></floop_desc>	Inserts the description of the current file.
<floop_density></floop_density>	Inserts the default key density of the current file.
<floop_name></floop_name>	Inserts the file name of the current file definition.
<floop_name_noext></floop_name_noext>	Inserts the file name of the current file definition, with the file extension removed.
<floop_odbc_name></floop_odbc_name>	Inserts the ODBC table name of the file definition that is currently being processed.
<floop_pagesize></floop_pagesize>	Inserts the page size of the current file.
<floop_rectype></floop_rectype>	Inserts a string indicating the record type of the current file.
<floop_rps_name></floop_rps_name>	Inserts the name of the repository file definition that is currently being processed.
<floop_static_rfa></floop_static_rfa>	Inserts a string indicating whether the current file has static RFA's enabled.

FLOOP_ADDRESSING

Inserts the addressing size of the current data file.

Usage

<FLOOP_ADDRESSING>

Possible values

- 32
- 40

Alternate forms

None

See also

None

FLOOP_CHANGE_TRACKING

Inserts a string indicating whether the structures first assigned file has change tracking enabled.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

NOTE: Supported in Synergy/DE V10.1 and higher.

Usage

<FLOOP_CHANGE_TRACKING>

Possible values

- YES
- NO

Alternate forms

<floop_change_tracking>

See also

Other structure tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FLOOP_COMPRESSION

Inserts a string indicating whether the current file has data compression enabled.

Usage

<FLOOP_COMPRESSION>

Possible values

- YES
- NO

Alternate forms

<floop_compression>

See also

None

FLOOP_DESC

Inserts the description of the current file.

Usage

<FLOOP_DESC>

Alternate forms

None

See also

None

FLOOP_DENSITY

Inserts the default key density of the current file.

Usage

<FLOOP_DENSITY>

Possible values

- 50
- 100

Alternate forms

None

See also

None

FLOOP_NAME

Inserts the file name of the current file definition.

Usage

<FLOOP_NAME>

Alternate forms

None

See also

• <FLOOP_NAME_NOEXT>

FLOOP_NAME_NOEXT

Inserts the file name of the current file definition, with the file extension removed.

This token can be useful when constructing file specifications where the file extension is based on a variable company code or other data.

Usage

<FLOOP_NAME_NOEXT>

Alternate forms

None

See also

<FLOOP_NAME>

FLOOP_ODBC_NAME

Inserts the ODBC table name of the file definition that is currently being processed. If no ODBC table name is specified then the name of the first assigned structure is used.

Usage

<FLOOP_ODBC_NAME>

Example Output

CUSTOMER

Alternate forms

- <floop_odbc_name>
- <Floop_Odbc_Name>
- <Floop_odbc_name>
- <FloopOdbcName>
- <floopOdbcName>

See also

•

FLOOP_PAGESIZE

Inserts the page size of the current file.



<FLOOP_PAGESIZE>

Possible values

- 512
- 1024
- 2048
- 4096
- 8192
- 16384
- 32768

NOTE: Page sizes 16384 and 32768 were introduced with ISAM Rev 6 in Synergy/DE 10.1

Alternate forms

None

See also

None

FLOOP_RECTYPE

Inserts a string indicating the record type of the current file.

Usage

<FLOOP_RECTYPE>

Possible values

- FIXED
- VARIABLE
- MULTIPLE

Alternate forms

<floop_rectype>

See also

None

FLOOP_RPS_NAME

Inserts the name of the repository file definition that is currently being processed.

Usage

<FLOOP_RPS_NAME>

Example Output

CUSTOMER

Alternate forms

- <floop_rps_name>
- <Floop_Rps_Name>
- <Floop_rps_name>
- <FloopRpsName>
- <floopRpsName>

See also

•

FLOOP_STATIC_RFA

Inserts a string indicating whether the current file has static RFA's enabled.

Usage

<FLOOP_STATIC_RFA>

Possible values

- YES
- NO

Alternate forms

<floop_static_rfa>

See also

None

FLOOP_STORED_GRFA

Inserts a string indicating whether the structures first assigned file has stored GRFA's enabled.

CodeGen will display an error if it encounters this token and the structure being processed has not been assigned to a repository file definition.

NOTE: Supported with Synergy/DE V10.1 and higher.

Usage

<FLOOP_STORED_GRFA>

Possible values

- YES
- NO

Alternate forms

<floop_stored_grfa>

See also

Other structure tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

FLOOP_TYPE

Inserts a string indicating the file type of the current file.

Usage

<FLOOP_TYPE>

Possible values

- ASCII
- DBL ISAM
- RELATIVE
- USER DEFINED

Alternate forms

None

See also

None

FLOOP_UTEXT

Inserts the user text string of the current file.

Usage

<FLOOP_UTEXT>

Alternate forms

None

See also

None

File Loop Expression Tokens

File loop expressions are expressions that can only be used within the context of a file loop and allow you to conditionally include or exclude code based on the attributes of the current file.

File Loop Expression	Description
<if ascii=""></if>	The file is an ASCII file.
<if change_tracking=""></if>	The file has change tracking enabled. (Synergy 10.1 and higher).
<if description=""></if>	The file has a file description specified.
<if isam=""></if>	The file is an ISAM file.
<if nochange_tracking=""></if>	The file does not have change tracking enabled. (Synergy 10.1 and higher).
<if nodescription=""></if>	The file does not have a file description specified.
<if NORECORDCOMPRESSION></if 	The file does not have record compression enabled.
<if nostored_grfa=""></if>	The file does not have stored GRFA enabled. (Synergy 10.1 and higher).
<if notascii=""></if>	The file is not an ASCII file.
<if notisam=""></if>	The file is not an ISAM file.
<if notrecordtypefixed=""></if>	The file does not contain fixed length records.
<if NOTRECORDTYPEVARIABLE></if 	The file does not contain variable length records.

<if NOTRECORDTYPEMULTIPLE></if 	The file does not contain multiple fixed length records.
<if notrelative=""></if>	The file is not a RELATIVE file.
<if notstaticrfa=""></if>	The file does not use static RFAs.
<if notterabyte=""></if>	The file is not a terabyte file.
<if notuserdefined=""></if>	The file is not a user-defined file type.
<if nousertext=""></if>	The file does not have user defined text specified.
<if pagesize512=""></if>	The page size of the file is 512 bytes.
<if pagesize1024=""></if>	The page size of the file is 1024 bytes.
<if pagesize2048=""></if>	The page size of the file is 2048 bytes.
<if pagesize4096=""></if>	The page size of the file is 4096 bytes.
<if pagesize8192=""></if>	The page size of the file is 8192 bytes.
<if pagesize16384=""></if>	The page size of the file is 16384 bytes.
<if pagesize32768=""></if>	The page size of the file is 32768 bytes.
<if recordcompression=""></if>	The file has record compression enabled.
<if recordtypefixed=""></if>	The file contains fixed length records.
<if recordtypevariable=""></if>	The file contains variable length records.
<if recordtypemultiple=""></if>	The file contains multiple fixed length records.
<if relative=""></if>	The file is a RELATIVE file.
<if staticrfa=""></if>	The file uses static RFAs.

The file has stored GRFA enabled.
<IF STORED_GRFA> (Synergy 10.1 and higher).

<IF TERABYTE> The file is a terabyte file.

<IF USERDEFINED> The file is a user-defined file type.

<IF USERTEXT>
The file has user defined text specified.

Tag Loops

A tag loop is a template file construct which allows you to iterate through the collection of a tags that CodeGen has information about. In order to use tag loops you must be generating code based on a repository structure that has at least one field tag defined.

Tag loops are delimited by a matching pair of **TAG_LOOP** and **TAG_LOOP** tags that surround the template code to be inserted for each tag. The code between the opening and closing tags is repeated for each tag in the structure.

Tag loop tokens are tokens that can only be used within a tag loop:

- Tag Loop Expansion Tokens
- Tag Loop Expression Tokens

There are two types of tag loops, called multi-line tag loops, and in-line tag loops.

Multi-Line Tag Loops

Multi-line tag loops occur when the opening and closing tag loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
[code]

<TAG_LOOP>

code

</TAG_LOOP>

[code]
```

Multi-line tag loops generate one or more lines of output code for each tag that is processed.

In-Line Tag Loops

In-line tag loops exist when the opening and closing tag loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

In-line tag loops generate code into the current output line.

Tag Loop Expansion Tokens

Tag loop expansion tokens are used to insert information about the current tag being processed within the context of a tag loop into the output stream. Tag loop tokens can only be used inside a tag loop.

Tag Loop Token	Description
<tagloop_connector_c></tagloop_connector_c>	For multi-expression tags, inserts the C-style operator that is used to connect the current tag expression with the previous tag expression.
<tagloop_connectio_dbl></tagloop_connectio_dbl>	For multi-expression tags, inserts the DBL-style operator that is used to connect the current tag expression with the previous tag
<tagloop_field_altname></tagloop_field_altname>	expression. Inserts the tag fields alternate name. Refer to <field_altname> for additional information.</field_altname>
<tagloop_field_basename></tagloop_field_basename>	Inserts the tag fields base name. Refer to <field_basename> for additional information.</field_basename>
<tagloop_field_name></tagloop_field_name>	Inserts the tag fields name.
<tagloop_field_odbcname></tagloop_field_odbcname>	Inserts the tag fields ODBC compatible name. Refer to <field_odbcname> for additional information.</field_odbcname>
<tagloop_field_originalname></tagloop_field_originalname>	Inserts the tag fields original name.
<tagloop_field_sqlname></tagloop_field_sqlname>	Inserts the tag fields SQL compatible name. Refer to <field_sqlname> for</field_sqlname>

<tagloop_operator_c></tagloop_operator_c>	additional information. Inserts the appropriate C operator to evaluate the tag field against the tag value.
<tagloop_operator_dbl></tagloop_operator_dbl>	Inserts the appropriate DBL operator to evaluate the tag field against the tag value.
<tagloop_sequence></tagloop_sequence>	Inserts the sequence number of the current tag within the collection of tags.
<tagloop_tag_name></tagloop_tag_name>	Inserts the name of the current tag.
<tagloop_tag_value></tagloop_tag_value>	Inserts the comparison value of the current tag.

TAGLOOP_CONNECTOR_C

For multi-expression tags, inserts the C-style operator that is used to connect the current tag expression with the previous tag expression. The token is replaced by a blank string for the first tag expression.

Usage

<TAGLOOP_CONNECTOR_C>

Possible values

- &&
- |

Alternate forms

None

- Introduction to tag loops
- <TAGLOOP_CONNECTOR_DBL>
- Counter tokens
- Loop utility tokens

TAGLOOP_CONNECTOR_DBL

For multi-expression tags, inserts the DBL-style operator that is used to connect the current tag expression with the previous tag expression. The token is replaced by a blank string for the first tag expression.

Usage

<TAGLOOP_CONNECTOR_DBL>

Possible values

- .AND.
- .OR.

Alternate forms

<tagloop_connector_dbl>

- Introduction to tag loops
- <TAGLOOP_CONNECTOR_C>
- Counter tokens
- Loop utility tokens

TAGLOOP FIELD ALTNAME

Inserts the tag fields alternate name. Refer to <FIELD_ALTNAME> for additional information.

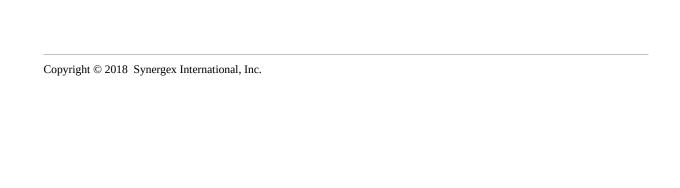
Usage

<TAGLOOP_FIELD_ALTNAME>

Alternate forms

- <tagloop_field_altname>
- <Tagloop_Field_Altname>
- <Tagloop_field_altname>
- <TagloopFieldAltname>
- <tagloopFieldAltname>

- Introduction to tag loops
- <TAGLOOP_FIELD_BASENAME>
- <TAGLOOP_FIELD_NAME>
- <TAGLOOP_FIELD_ODBCNAME>
- <TAGLOOP_FIELD_ORIGINALNAME>
- <TAGLOOP_FIELD_SQLNAME>
- Counter tokens
- Loop utility tokens



TAGLOOP_FIELD_BASENAME

Inserts the tag fields base name. Refer to <FIELD_BASENAME> for additional information.

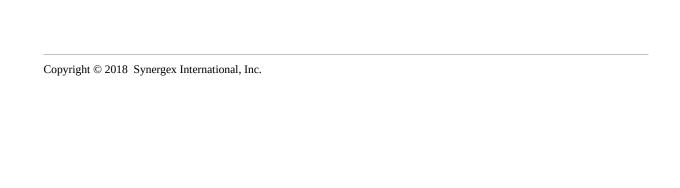
Usage

<TAGLOOP_FIELD_BASENAME>

Alternate forms

- <tagloop_field_basename>
- <Tagloop_Field_Basename>
- <Tagloop_field_basename>
- <TagloopFieldBasename>
- <tagloopFieldBasename>

- Introduction to tag loops
- <TAGLOOP_FIELD_ALTNAME>
- <TAGLOOP_FIELD_NAME>
- <TAGLOOP_FIELD_ODBCNAME>
- <TAGLOOP_FIELD_ORIGINALNAME>
- <TAGLOOP_FIELD_SQLNAME>
- Counter tokens
- Loop utility tokens



TAGLOOP_FIELD_CSTYPE

Inserts a string indicating the C# equivalent data type of the tag field currently being processed.

Usage

<TAGLOOP_FIELD_CSTYPE>

Possible values

- bool
- byte
- double
- float
- int
- long
- decimal
- DateTime
- Nullable<DateTime>
- sbyte
- short
- string
- uint
- ulong
- ushort

Alternate forms

None

See also

• Other tag loop expansion tokens

TAGLOOP_FIELD_NAME

Inserts the tag fields name.

Usage

<TAGLOOP_FIELD_NAME>

Alternate forms

- <tagloop_field_name>
- <Tagloop_Field_Name>
- <Tagloop_field_name>
- <TagloopFieldName>
- <tagloopFieldName>

- Introduction to tag loops
- <TAGLOOP_FIELD_ALTNAME>
- <TAGLOOP_FIELD_BASENAME>
- <TAGLOOP_FIELD_ODBCNAME>
- <TAGLOOP_FIELD_ORIGINALNAME>
- <TAGLOOP_FIELD_SQLNAME>
- Counter tokens
- Loop utility tokens

TAGLOOP_FIELD_ODBCNAME

Inserts the tag fields ODBC compatible name. Refer to <FIELD ODBCNAME> for additional information.

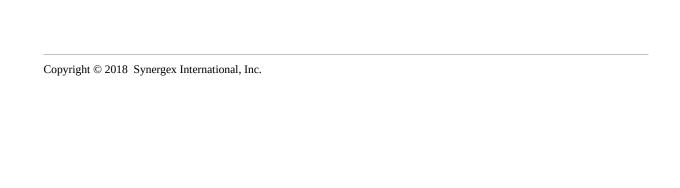
Usage

<TAGLOOP_FIELD_ODBCNAME>

Alternate forms

- <tagloop_field_odbcname>
- <Tagloop_Field_Odbcname>
- <Tagloop_field_odbcname>
- <TagloopFieldOdbcname>
- <tagloopFieldOdbcname>

- Introduction to tag loops
- <TAGLOOP_FIELD_ALTNAME>
- <TAGLOOP_FIELD_BASENAME>
- <TAGLOOP_FIELD_NAME>
- <TAGLOOP_FIELD_ORIGINALNAME>
- <TAGLOOP_FIELD_SQLNAME>
- Counter tokens
- Loop utility tokens



TAGLOOP FIELD ORIGINALNAME

Inserts the tag fields original name. Refer to <FIELD ORIGINAL NAME> for additional information.

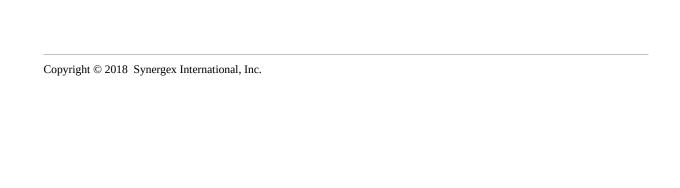
Usage

<TAGLOOP_FIELD_ORIGINALNAME>

Alternate forms

- <tagloop_field_originalname>
- <Tagloop_Field_Originalname>
- <Tagloop_field_originalname>
- <TagloopFieldOriginalname>
- <tagloopFieldOriginalname>

- Introduction to tag loops
- <TAGLOOP_FIELD_ALTNAME>
- <TAGLOOP_FIELD_BASENAME>
- <TAGLOOP_FIELD_NAME>
- <TAGLOOP_FIELD_ODBCNAME>
- <TAGLOOP_FIELD_SQLNAME>
- Counter tokens
- Loop utility tokens



TAGLOOP_FIELD_SQLNAME

Inserts the tag fields SQL compatible name. Refer to <FIELD SQLNAME> for additional information.

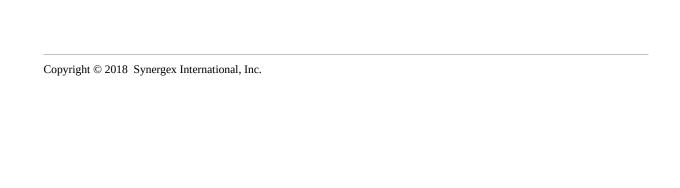
Usage

<TAGLOOP_FIELD_SQLNAME>

Alternate forms

- <tagloop_field_sqlname>
- <Tagloop_Field_Sqlname>
- <Tagloop_field_sqlname>
- <TagloopFieldSqlname>
- <tagloopFieldSqlname>

- Introduction to tag loops
- <TAGLOOP_FIELD_ALTNAME>
- <TAGLOOP_FIELD_BASENAME>
- <TAGLOOP_FIELD_NAME>
- <TAGLOOP_FIELD_ODBCNAME>
- <TAGLOOP_FIELD_ORIGINALNAME>
- Counter tokens
- Loop utility tokens



TAGLOOP_OPERATOR_C

Inserts the appropriate C operator to evaluate the tag field against the tag value.

Usage

<TAGLOOP_OPERATOR_C>

Possible values

- ==
- >=
- >
- <=
- <
- !=

Alternate forms

None

- Introduction to tag loops
- <TAGLOOP_OPERATOR_DBL>
- Counter tokens
- Loop utility tokens

TAGLOOP_OPERATOR_DBL

Inserts the appropriate DBL operator to evaluate the tag field against the tag value.

Usage

<TAGLOOP_OPERATOR_DBL>

Possible values

- .EQ.
- .GE.
- .GT.
- .LE.
- .LT.
- .NE.

Alternate forms

<tagloop_operator_dbl>

- Introduction to tag loops
- <TAGLOOP_OPERATOR_C>
- Counter tokens
- Loop utility tokens

TAGLOOP_SEQUENCE

Inserts the sequence number of the current tag within the collection of tags.

Usage

<TAGLOOP_SEQUENCE>

Alternate forms

None

See also

- Introduction to tag loops
- Counter tokens
- Loop utility tokens

TAGLOOP_TAG_NAME

Inserts the name of the current tag.

Usage

<TAGLOOP_TAG_NAME>

Alternate forms

None

See also

- Introduction to tag loops
- Counter tokens
- Loop utility tokens

TAGLOOP_TAG_VALUE

Inserts the comparison value of the current tag.

Usage

<TAGLOOP_TAG_VALUE>

Alternate forms

None

See also

- Introduction to tag loops
- Counter tokens
- Loop utility tokens

Tag Loop Expression Tokens

Tag loop expressions are expressions that can only be used within the context of a tag loop and allow you to conditionally include or exclude code based on the attributes of the current tag.

Tag Loop Expression	Description
<if comparison_eq=""></if>	The tag includes an equal to comparison.
<if comparison_ge=""></if>	The tag includes a greater than or equal to comparison.
<if comparison_gt=""></if>	The tag includes a greater than comparison.
<if comparison_le=""></if>	The tag includes a less than or equal to comparison.
<if comparison_lt=""></if>	The tag includes a less than comparison.
<if comparison_ne=""></if>	The tag includes a not equal to comparison.
<if COMPARISON_NOT_EQ></if 	The tag does not include an equal to comparison.
<if COMPARISON_NOT_GE></if 	The tag does not include a greater than or equal to comparison.
<if COMPARISON_NOT_GT></if 	The tag does not include a greater than comparison.
<if COMPARISON_NOT_LE></if 	The tag does not include a less than or equal to comparison.
<if COMPARISON_NOT_LT></if 	The tag does not include a less than comparison.
<if COMPARISON_NOT_NE></if 	The tag does not include a not equal to comparison.

<if connector_none=""></if>	The tag is not linked to a previous tag because it is the first or only tag.
<if connector_and=""></if>	The tag is linked to a previous tag with an AND operator.
<if connector_or=""></if>	The tag is linked to a previous tag with an OR operator.
<if CONNECTOR_NOT_NONE</if 	The tag is linked to a previous tag with an AND or an OR operator.
<if CONNECTOR_NOT_AND></if 	The tag is not linked to a previous tag with an AND operator.
<if CONNECTOR_NOT_OR></if 	The tag is not linked to a previous tag with an OR operator.
<if multiple_tags=""></if>	The structure being processed has multiple tags.
<if single_tag=""></if>	The structure being processed has a single tag.

Structure Loops

A structure loop is a template file construct which allows you to iterate through the collection of structures that CodeGen has information about. In order to use structure loops you must be generating code based on multiple repository structures at the same time.

CodeGen can process multiple structures in one of two ways:

- Multiple structures can be specified on the command line, after the -s command line option, and the -ms command line option can be used to specify that all of the structures are to be processed together.
- The -file command line option can be used to use a repository file definition as the starting point for code generation, and that file definition can have multiple structures assigned.

Structure loops are delimited by a matching pair of <STRUCTURE_LOOP> and </STRUCTURE_LOOP> tags that surround the template code to be inserted for each structure. The code between the opening and closing tags is repeated for each structure being processed. This code may contain any other tokens.

If a structure loop is encountered in a template when only a single structure is being processed then an error will be generated and CodeGen will stop processing.

Structure loops can contain code that is to be repeated for each structure being processed, and can also contain any other tokens that are valid when processing a single structure. CodeGen simply repeats the code between the opening and closing structure loop tokens the appropriate number of times based on the number of structures being processed, switching structure context as it goes.

There are two types of structure loops, called multi-line structure loops, and in-line structure loops.

Multi-Line Structure Loops

Multi-line structure loops occur when the opening and closing structure loop

tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
[code]
<STRUCTURE_LOOP>
code
</STRUCTURE_LOOP>
[code]
```

Multi-line structure loops generate one or more lines of output code for each structure that is processed.

In-Line Structure Loops

In-line structure loops exist when the opening and closing structure loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
[code] STRUCTURE_LOOP> code STRUCTURE_LOOP> [code]
```

In-line structure loops generate code into the current output line.

See also

<STRUCTURE#n>

Structure Loop Expansion Tokens

CodeGen supports structure loop expansion tokens, but none have been defined yet. Loop utility expansion tokens can also be used within structure loops.

Structure Loop Expression Tokens

CodeGen supports structure loop expression tokens, but none have been defined yet. Loop utility expressions can also be used within structure loops.

Loop Utility Tokens

Loop utility tokens are special tokens that can be used in any of the loop structures; they expose or allow you to test information about the progress of the loop itself.

- Loop utility expansion tokens
- Loop utility expression tokens

Loop Utility Expansion Tokens

Loop utility expansion tokens are tokens that are available for use inside any loop construct, and insert information based on the processing status of the current loop.

Loop Utility Token	Description
<,>	Inserts a comma UNLESS the final loop iteration is being processed, in which case inserts nothing.
<,AND> <,and>	Inserts nothing if the final loop iteration is being performed. Inserts the text "AND" if the last but one iteration is being performed. Otherwise inserts a comma and a space. This token can be useful when producing documentation, as it allows you to generate text like "The primary colors are red, green and blue."
<,OR> <,or>	Inserts nothing if the final loop iteration is being performed. Inserts the text "OR" if the last but one iteration is being performed. Otherwise inserts a comma and a space. This token can be useful when producing documentation, as it allows you to generate text like "Select a primary color red, green or blue."
<+>	Inserts a plus sign UNLESS the final loop iteration is being processed, in which case inserts nothing.

< <u>:</u> >	Inserts a colon UNLESS the final loop iteration is being processed, in which case inserts nothing.
<&&>	Inserts a logical AND operator (&&) UNLESS the final loop iteration is being processed, in which case inserts nothing.
<.AND.>	Inserts a Synergy logical AND operator (.AND.) UNLESS the final loop iteration is being processed, in which case inserts nothing.
<and></and>	Inserts the word AND UNLESS the final loop iteration is being processed, in which case inserts nothing.
< >	Inserts a pipe character () UNLESS the final loop iteration is being processed, in which case inserts nothing.
< >	Inserts a logical OR operator () UNLESS the final loop iteration is being processed, in which case inserts nothing.
<.OR.>	Inserts a Synergy logical OR operator (.OR.) UNLESS the final loop iteration is being processed, in which case inserts nothing.
<or></or>	Inserts the word OR, UNLESS the final loop iteration is being processed, in which case inserts nothing.

<BSLASH>

<FSLASH>

<PROCESSED EXCLUSIVE>

<PROCESSED INCLUSIVE>

<REMAINING EXCLUSIVE>

Inserts a backward slash character (\) UNLESS the final loop iteration is being processed, in which case inserts nothing.

Inserts a forward slash character (/) UNLESS the final loop iteration is being processed, in which case inserts nothing.

Inserts the number of items already processed by the current loop EXCLUDING the current item. For example, if a loop with 10 items is being processed, and the 8th item is currently being processed, the value 7 will be inserted. This is because 7 items have already been processed before the current item.

Inserts the number of items already processed by the current loop INCLUDING the current item. For example, if a loop with 10 items is being processed, and the 8th item is currently being processed, the value 8 will be inserted. This is because the loop is currently processing the 8th item.

Inserts the number of remaining items to be processed by the current loop EXCLUDING the current item. For example, if a loop with 10 items is being processed, and the 8th item is currently being processed, the value 2 will be inserted. This is because there are 2 items still to be processed after the current item.

Similar to <REMAINING EXCLUSIVE> <REMAINING_EXCLUSIVE_MAX_n> except that if the number of items remaining is greater than n (a positive integer value) then the value of *n* will be inserted.

<REMAINING_INCLUSIVE>

Inserts the number of remaining items to be processed by the current loop INCLUDING the current item. For example, if a loop with 10 items is being processed, and the 8th item is currently being processed, the value 3 will be inserted. This is because there are 3 items still to be processed, including the current field.

<REMAINING_INCLUSIVE_MAX_n>

Similar to <REMAINING_INCLUSIVE> except that if the number of items remaining is greater than n (a positive integer value) then the value of *n* will be inserted.

<TOTAL_ITEMS>

Inserts the total number of items that will be processed by the current loop.

Loop Utility Expression Tokens

Loop utility expression tokens are expressions that are available for use inside any loop construct, and allow you to test things relating to the status of the current loop.

Loop Utility Expression	Description		
<if first=""></if>	The first iteration of the loop is currently being processed.		
<if last=""></if>	The last iteration of the loop is currently being processed.		
<if more=""></if>	There are more loop iterations to be processed.		
<if nomore=""></if>	There are no more loop iterations to be processed (same as <if last="">).</if>		
	Evaluates the number of items already processed by a loop (EXCLUDING the item currently being processed) using a comparison operator and numeric literal value represented here by <i>n</i> . Possible values for op are:		
<if PROCESSED_EXCLUSIVE_op_n></if 	EQ Equal to NE Not equal to LT Less than GT Greater than LE Less than or equal to GE Greater than or equal to DB Divisible by		
	An example of using this expression is:		
	<if PROCESSED_EXCLUSIVE_EQ_10></if 		

template code </IF>

<IF PROCESSED_INCLUSIVE_op_n>

Evaluates the number of items already processed by a loop (INCLUDING the item currently being processed) using a comparison operator and numeric literal value. For more information on how to use this expression refer to <IF PROCESSED EXCLUSIVE op n>.

<IF

Evaluates the number of items still to be processed by a loop (EXCLUDING the item currently being processed) using a REMAINING_EXCLUSIVE_ op_n comparison operator and numeric literal value. For more information on how to use this expression refer to <IF PROCESSED_EXCLUSIVE_op_n>.

<IF REMAINING INCLUSIVE op n>

Evaluates the number of items still to be processed by a loop (INCLUDING the item currently being processed) using a comparison operator and numeric literal value. For more information on how to use this expression refer to <IF PROCESSED_EXCLUSIVE_op_n>.

<IF TOTAL_ITEMS_op_n>

Evaluates the total number of items that will be processed by a loop (INCLUDING the item currently being processed) using a comparison operator and numeric literal value. For more information on how to use this expression refer to <IF PROCESSED EXCLUSIVE op n>.

Not In Loop Tokens

Not in loop tokens are a special class of token that can be used anywhere in a template file EXCEPT in a loop.

CodeGen includes support for both not in loop expansion and expression tokens, but none have been defined yet.

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

Processing UI Toolkit Windows

CodeGen can process some metadata that originates from UI Toolkit Input Window definitions located in UI Toolkit window script files. This functionality is enabled via the -w command line option.

This feature allows you to generate basic UI in the form of forms for windows, web, or mobile applications.

Window Tokens are tokens that can only be used when processing a UI Toolkit input window.

• Window Expansion Tokens

Window Expansion Tokens

Window expansion tokens insert information about the UI Toolkit input window that is currently being processed. They can only be used when processing a UI Toolkit input window via the -w command line option.

Window Expansion Token	Description
<window_height></window_height>	Inserts the height of the input window in rows.
<window_heightpx></window_heightpx>	Inserts the height of the input window in pixels.
<window_name></window_name>	If code generation is being performed based on the definition of a UI Toolkit input window in a window script file then inserts the name of the input window. Otherwise the token inserts the
<window_width></window_width>	name of the repository structure being processed. Inserts the width of the input window in screen columns.
<window_widthpx></window_widthpx>	Inserts the width of the input window in pixels.

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

WINDOW_HEIGHT

Inserts the height of the input window in rows.

Usage

<WINDOW_HEIGHT>

Alternate forms

None

See also

None

WINDOW_HEIGHTPX

Inserts the height of the input window in pixels. If code generation is based on an existing UI Toolkit window then the height of the source window is used to calculate the value, otherwise the number of fields in the structure is used.

Calculations are based on a default of 25 pixels per row, and this can be overridden using the -ch command line option.

Usage

<WINDOW_HEIGHTPX>

Alternate forms

None

See also

None

WINDOW_NAME

If code generation is being performed based on the definition of a UI Toolkit input window in a window script file then inserts the name of the input window. Otherwise the token inserts the name of the repository structure being processed.

Usage

<WINDOW_NAME>

Alternate forms

- <window_name>
- <Window_name>
- <Window_Name>
- <WindowName>
- <windowName>

See also

None

WINDOW_WIDTH

Inserts the width of the input window in screen columns.

Usage

<WINDOW_WIDTH>

Alternate forms

None

See also

None

WINDOW_WIDTHPX

Inserts the width of the input window in pixels. A default character width of 12 pixels per column is used, and this can be overridden using the -cw command line option.

Usage

<WINDOW_WIDTHPX>

Alternate forms

None

See also

None

Processing a Synergy Method Catalog

If your Synergy application development environment includes the use of Synergy/DE xfServerPlus then you can generate code based on the metadata contained within the Synergy Method Catalog. To enable this feature you must export your method catalog to an XML file, and then use the -smc and optionally the -interface command line options.

Processing a Single Interface

To process a single interface you must use both the -smc and -interface options together. For example:

codegen -smc C:\my\files\smc.xml -interface MyInterface -t template -r

Within a template file you can use a method loop to iterate through the methods that are members of the interface being processed, and within the method loop you can use method loop expansion tokens and method loop expression tokens.

<METHOD_LOOP>

;Method loop code and tokens goes here

</METHOD_LOOP>

If a method has parameters you can iterate through them using a parameter loop, and within the parameter loop you can use parameter loop expansion tokens and parameter loop expression tokens.

<METHOD_LOOP>

;Method loop code and tokens goes here

<PARAMETER_LOOP>

;Parameter loop code and tokens goes here

</PARMAETER_LOOP>

;Method loop code and tokens goes here

</METHOD_LOOP>

Processing All Interfaces

</INTERFACE_LOOP>

If you wish to process all of the interfaces that are defined in the method catalog, one at a time, then use the -smc option alone, and use an interface loop in your template file. Inside the interface loop you can use interface loop expansion tokens and interface loop expression tokens. Interface loops can contain method loops and parameter loops:

```
<INTERFACE_LOOP>

;Interface loop code and tokens goes here

<METHOD_LOOP>

;Method loop code and tokens goes here

<PARAMETER_LOOP>

;Parameter loop code and tokens goes here

</PARMAETER_LOOP>

;Method loop code and tokens goes here

</METHOD_LOOP>

;Interface loop code and tokens goes here
```

Interface Loops

An interface loop is a template file construct which allows you to iterate through the collection of interfaces that CodeGen has information about. These interface definitions come from a Synergy Method Catalog.

Interface loops are delimited by a matching pair of

<INTERFACE_LOOP> and </INTERFACE_LOOP> tags, which surround the template code to be inserted for each interface. The code between the opening and closing tags of an interface loop is repeated for each interface in the method catalog being processed. This code may contain other generic and structure tokens, and may also contain special interface loop tokens, which are discussed later. Loop utility tokens can also be used in interface loops.

There are two types of interface loops, called multi-line interface loops, and in-line interface loops.

Multi-Line Interface Loops

Multi-line interface loops occur when the opening and closing interface loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

[code]

<INTERFACE_LOOP>

</INTERFACE_LOOP>

[code]

Multi-line interface loops generate one or more lines of output code for each interface that is processed.

In-Line Interface Loops

In-line interface loops exist when the opening and closing interface loop tags appear on the same line in a template file, and delimit part of a line of

template code, like this:

[code] **<INTERFACE_LOOP>** code **</INTERFACE_LOOP>** [code]

In-line interface loops generate code into the current output line.

Interface Loop Expansion Tokens

Interface loop expansion tokens can be used to inject information about the interface being processed into the code output stream.

	Description
<pre><intedeace methods=""></intedeace></pre>	The number of methods within the current interface.
\INTERFACE_WETHODS	interface.
<interface_name></interface_name>	The name of the current interface.

INTERFACE_METHODS

Inserts the number of methods within the current interface.

Usage

<INTERFACE_METHODS>

Possible values

• A positive integer.

Alternate forms

None

Example of use

See also

• Other interface loop expansion tokens

INTERFACE_NAME

Inserts the name of the current interface.

Usage

<INTERFACE_NAME>

Possible values

• Interface name.

Alternate forms

None

Example of use

See also

• Other interface loop expansion tokens

Interface Loop Expression Tokens

CodeGen has support for interface loop expression tokens, but none have been created yet.

Method Loops

A method loop is a template file construct which allows you to iterate through the collection of methods that CodeGen has information about. These method definitions come from an interface definition in a Synergy Method Catalog.

Method loops are delimited by a matching pair of **METHOD_LOOP**> and **</METHOD_LOOP**> tags, which surround the template code to be inserted for each method. The code between the opening and closing tags of an method loop is repeated for each method in the interface catalog being processed. This code may contain other generic and structure tokens, and may also contain special method loop tokens, which are discussed later. Loop utility tokens can also be used in method loops.

There are two types of method loop, called multi-line method loops, and inline method loops.

Multi-Line Method Loops

Multi-line method loops occur when the opening and closing method loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

[code]

<METHOD_LOOP>code </METHOD_LOOP>

[code]

Multi-line method loops generate one or more lines of output code for each method that is processed.

In-Line method Loops

In-line method loops exist when the opening and closing method loop tags appear on the same line in a template file, and delimit part of a line of template code, like this:

[code] **<METHOD_LOOP>** code **</METHOD_LOOP>** [code]

In-line method loops generate code into the current output line.

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

Method Loop Expansion Tokens

Method loop expansion tokens can be used to inject information about the method being processed into the code output stream.

Method Loop Token	Description
<method_comment></method_comment>	The comment associated with the method.
<method_encrypt></method_encrypt>	Whether the method requires encryption.
<method_elb></method_elb>	The file spec of the ELB containing the method.
<method_id></method_id>	The unique method ID of the method.
<method_name></method_name>	The name of the method.
<method_parameters></method_parameters>	The number of parameters defined in the method.
<method_return_coerce_type></method_return_coerce_type>	The coerce type of the method return value.
<method_return_comment></method_return_comment>	The commend associated with the return value.
<method_return_date_format< td=""><td>The date format associated with the return value.</td></method_return_date_format<>	The date format associated with the return value.
<method_return_enum></method_return_enum>	The enumeration format associated with the return value.
<method_return_precision></method_return_precision>	The decimal precision of the return value.
<method_return_size></method_return_size>	The size of the return value.
<method_return_type></method_return_type>	The data type of the return value.
<method_routine></method_routine>	The name of the underlying Synergy routine.
<method_type></method_type>	The type of method.

METHOD_COMMENT

Inserts the comment associated with the current method.

Usage

<METHOD_COMMENT>

Possible values

• String (may be empty)

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_ENCRYPT

Inserts a boolean value indicating whether the current method requires overthe-wire encryption.

Usage

<METHOD_ENCRYPT>

Possible values

- True
- False

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_ELB

Inserts the file specification of the ELB containing the current method.

Usage

<METHOD_ELB>

Possible values

• File specification.

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_ID

Inserts the unique ID of the current method.

Usage

<METHOD_ID>

Possible values

• Method ID string.

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_NAME

Inserts the name of the current method.

Usage

<METHOD_NAME>

Possible values

• Method name.

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_PARAMETERS

Inserts the number of parameters that the current method has/

Usage

<METHOD_PARAMETERS>

Possible values

• Integer greater than or equal to zero.

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_RETURN_COERCE_TYPE

Inserts the coerce type associated with the current methods return value.

Usage

<method_return_coerce_type>

Possible values

- None
- Boolean
- Byte
- DateTime
- DateTimeNullable
- Decimal
- DecimalNullable
- Double
- Float
- Int
- Long
- Sbyte
- Short
- Uint
- Ulong
- Ushort

Alternate forms

None

Example of use

See also

• Other m	ethod loop exp	ansion toker	ns	
Copyright © 2018 Syne	ergex International Inc			
copyright © 2010 Sync	igex international, inc.			

METHOD_RETURN_COMMENT

Inserts the comment associated with the current methods return value.

Usage

<METHOD_RETURN_COMMENT>

Possible values

• String (may be empty)

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_RETURN_DATE_FORMAT

Inserts the date format associated with the method return value.

Usage

<METHOD_RETURN_DATE_FORMAT>

Possible values

- None
- YYYYMMDD
- YYMMDD
- YYJJJ
- YYYYJJJ
- HHMM
- HHMMSS
- YYYYMMDDHHMISS

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_RETURN_ENUM

Inserts the name of the ENUM that defines the current methods return value. If the return type is not an enumeration then a blank string is returned.

Usage

<METHOD_RETURN_ENUM>

Possible values

• Enumeration name, or a blank string.

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_RETURN_PRECISION

Inserts the precision of the current methods return value, if the return value is implied decimal. If not a blank string is returned.

Usage

<METHOD_RETURN_PRECISION>

Possible values

• A positive integer, or a blank string.

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_RETURN_SIZE

Inserts the size of the current methods return value, if appropriate, or by a blank string if not.

Usage

<METHOD_RETURN_SIZE>

Possible values

• A positive integer or a blank string.

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_RETURN_TYPE

Inserts the return type of the current method.

Usage

<METHOD_RETURN_TYPE>

Possible values

- <blank string>
- A<size>
- D<size>
- D<size>.<precision>
- <enumName>
- I<size>
- ^VAL
- STRING

Alternate forms

<method_return_type>

Example of use

See also

Other method loop expansion tokens

METHOD_ROUTINE

Inserts the name of the synergy routine associated with the current method.

Usage

<METHOD_ROUTINE>

Possible values

• Routine name

Alternate forms

None

Example of use

See also

• Other method loop expansion tokens

METHOD_TYPE

Inserts the type of routine associated with the current method.

Usage

<METHOD_TYPE>

Possible values

- SUBROUTINE
- FUNCTION

Alternate forms

<method_type>

Example of use

See also

• Other method loop expansion tokens

Copyright $\ensuremath{\mathbb{C}}$ 2018 Synergex International, Inc.

Method Loop Expression Tokens

Method loop expressions are expressions that can only be used within the context of a method loop. CodeGen supports method loop expressions via the <**IF expression**> and </**IF [expression]**> tags. Note that loop utility expressions can also be used within method loops.

Expression	Description
<if comment=""></if>	The method has an associated comment.
<if encrypt=""></if>	The method requires over-the-wire encryption.
<if function=""></if>	The method is a function.
<if parameters=""></if>	The method has parameters.
<if alpha=""></if>	The method returns an alpha value.
<if coerce=""></if>	The return value is coerced to a different type.
<if coerce_boolean=""></if>	The return value is coerced to BOOLEAN.
<if coerce_byte=""></if>	The return value is coerced to BYTE.
<if coerce_datetime=""></if>	The return value is coerced to DATETIME.
<if COERCE_DATETIME_NULLABLE></if 	The return value is coerced to NULLABLE DATETIME.
<if coerce_decimal=""></if>	The return value is coerced to DECIMAL.
<if COERCE_DECIMAL_NULLABLE></if 	The return value is coerced to NULLABLE DECIMAL.
<if coerce_double=""></if>	The return value is coerced to

DOUBLE.

<IF COERCE_FLOAT>
The return value is coerced to FLOAT.

<IF COERCE_LONG>
The return value is coerced to LONG.

<IF COERCE_SBYTE> The return value is coerced to

SBYTE.

<IF COERCE_SHORT>
The return value is coerced to

SHORT.

<IF COERCE_ULONG> The return value is coerced to

ULONG.

<IF COERCE_USHORT>
The return value is coerced to

USHORT.

<IF HATVAL> The method returns a ^VAL (I4)

value.

<IF IMPLIED>
The method returns an implied

decimal value.

subroutine).

<IF SUBROUTINE>
The method is a subroutine.

Parameter Loops

A parameter loop is a template file construct which allows you to iterate through the collection of parameters of a method that CodeGen has information about. These parameter definitions come from a method definition in a Synergy parameter Catalog.

Parameter loops are delimited by a matching pair of <PARAMETER_LOOP> and </PARAMETER_LOOP> tags, which surround the template code to be inserted for each parameter. The code between the opening and closing tags of an parameter loop is repeated for each parameter in the interface catalog being processed. This code may contain other generic and structure tokens, and may also contain special parameter loop tokens, which are discussed later. Loop utility tokens can also be used in parameter loops.

There are two types of parameter loop, called multi-line parameter loops, and in-line parameter loops.

Multi-Line Parameter Loops

Multi-line parameter loops occur when the opening and closing parameter loop tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

[code]

<PARAMETER_LOOP>code </PARAMETER_LOOP>

[code]

Multi-line parameter loops generate one or more lines of output code for each parameter that is processed.

In-Line Parameter Loops

In-line parameter loops exist when the opening and closing parameter loop tags appear on the same line in a template file, and delimit part of a line of

template code, like this:

[code] **PARAMETER_LOOP** code **PARAMETER_LOOP** [code]

In-line parameter loops generate code into the current output line.

Processing Structure Parameters

When processing STRUCTURE parameters ONLY (check by using the parameter loop expression <IF STRUCTURE>) you may also use the following:

- Structure expansion tokens
- Structure expression tokens
- Field loops and field loop expansion and expression tokens.

An error will be generated if you attempt to use any of these when processing other types of parameter.

Parameter Loop Expansion Tokens

Parameter loop expansion tokens can be used to inject information about the parameter being processed into the code output stream.

Parameter Loop Token	Description
<parameter_coerce_type></parameter_coerce_type>	The coercion type of the current parameter.
<parameter_colection_type></parameter_colection_type>	The collection type of the current parameter.
<parameter_comment></parameter_comment>	The comment associated with the current parameter.
<parameter_data_table></parameter_data_table>	Whether the current parameter is exposed to .NET clients as a Data Table.
<parameter_date_format></parameter_date_format>	The date format of the current parameter.
<parameter_definition></parameter_definition>	The data type definition of the current parameter.
<parameter_direction></parameter_direction>	The direction of the current parameter.
<parameter_direction_pad></parameter_direction_pad>	The direction of the current parameter, padded to a length of 5 characters.
<parameter_enum></parameter_enum>	The name of the enum associated with the current parameter.
<parameter_name></parameter_name>	The name of the current parameter.
<parameter_pass_by></parameter_pass_by>	The passing mechanism of the current parameter.
<parameter_precision></parameter_precision>	The implied decimal precision of the current parameter.
<parameter_required></parameter_required>	The REQUIRED or OPTIONAL status of the current parameter.

<PARAMETER_SIZE>

The size of the current

parameter.

<PARAMETER_STRUCTURE>

The name of the

STRUCTURE associated with the current parameter.

Inserts the name of the

STRUCTURE that defines

the current parameters type.
If the name of the structure

does ends with an "S" then it

is removed.

Inserts the name of the STRUCTURE that defines the current parameters type. If the name of the structure

does not end with an "S" then one is added.

The data type of the current

parameter.

<PARAMETER_STRUCTURE_NOPLURAL>

<PARAMETER_STRUCTURE_PLURAL>

<PARAMETER_TYPE>

PARAMETER_COERCE_TYPE

Inserts the coercion type associated with the current parameter.

Usage

<PARAMETER_COERCE_TYPE>

Possible values

- None
- Boolean
- Byte
- DateTime
- DateTimeNullable
- Decimal
- DecimalNullable
- Double
- Float
- Int
- Long
- Sbyte
- Short
- Uint
- Ulong
- Ushort

Alternate forms

None

Example of use

See also

Other parameter loop expansion tokens
Copyright © 2018 Synergex International, Inc.

PARAMETER_COLLECTION_TYPE

Inserts the collection type of a collection parameter.

Usage

<PARAMETER_COLLECTION_TYPE>

Possible values

- None
- Array
- MemoryHandle
- ArrayList

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_COMMENT

Inserts the comment associated with the current parameter.

Usage

<PARAMETER_COMMENT>

Possible values

• String, could be blank.

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_DATA_TABLE

Inserts a boolean value indicating whether the current parameter is exposed to .NET clients as a DataTable.

Usage

<PARAMETER_DATA_TABLE>

Possible values

- True
- False

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_DATE_FORMAT

Inserts a string indicating the date format associated with the current date parameter.

Usage

<PARAMETER_DATE_FORMAT>

Possible values

- None
- YYYYMMDD
- YYMMDD
- YYJJJ
- YYYYJJJ
- HHMM
- HHMMSS
- YYYYMMDDHHMISS

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_DEFINITION

Inserts the parameter data type definition of the current parameter.

Usage

<PARAMETER_DEFINITION>

Possible values

- A<size>
- D<size>
- D<size>.<precision>
- D_HANDLE
- <enumName>
- I<size>
- STRING
- <structureName>

Alternate forms

<parameter_definition>

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_DIRECTION

Inserts the data flow direction associated with the current parameter.

Usage

<PARAMETER_DIRECTION>

Possible values

- IN
- OUT
- INOUT

Alternate forms

<parameter_direction>

Example of use

See also

- <PARAMETER_DIRECTION_PAD>
- Other parameter loop expansion tokens

PARAMETER_DIRECTION_PAD

Inserts the data flow direction associated with the current parameter. The string output will be right-padded with spaces to a length of 5 characters.

Usage

<PARAMETER_DIRECTION_PAD>

Possible values

- IN
- OUT
- INOUT

Alternate forms

<parameter_direction>

Example of use

See also

- <PARAMETER_DIRECTION>
- Other parameter loop expansion tokens

PARAMETER_ENUM

Inserts the name of the ENUM that defines the current parameters data type. If the return type is not an enumeration then a blank string is returned.

Usage

<PARAMETER_ENUM>

Possible values

• Enum name or a blank string.

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_NAME

Inserts the name of the current parameter.

Usage

<PARAMETER_NAME>

Possible values

• String.

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_PASS_BY

Inserts a string indicating the parameter passing mechanism of the current parameter.

Usage

<PARAMETER_PASS_BY>

Possible values

- DESCRIPTOR
- REFERENCE
- VALUE

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_PRECISION

Inserts the precision of the current parameter, if the return value is implied decimal. If not a blank string is returned.

Usage

<PARAMETER_PRECISION>

Possible values

• Positive integer or a blank string.

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_REQUIRED

Inserts a string indicating whether the current parameter is required or optional.

Usage

<PARAMETER_REQUIRED>

Possible values

- OPTIONAL
- REQUIRED

Alternate forms

<parameter_required>

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_SIZE

Inserts the size of the current parameter, if appropriate, or by a blank string if not.

Usage

<PARAMETER_SIZE>

Possible values

• A positive integer or a blank string.

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

PARAMETER_STRUCTURE

Inserts the name of the STRUCTURE that defines the current parameters type. If the return type is not a structure then a blank string is returned.

Usage

<PARAMETER_STRUCTURE>

Possible values

• String (structure name).

Alternate forms

- <parameter_structure>
- <Parameter_Structure>
- <Parameter_structure>
- <ParameterStructure>
- <parameterStructure>

Example of use

See also

- <PARAMETER_STRUCTURE_NOPLURAL>
- <PARAMETER_STRUCTURE_PLURAL>
- Other parameter loop expansion tokens

PARAMETER_STRUCTURE_NOPLURAL

Inserts the name of the STRUCTURE that defines the current parameters type. If the name of the structure does ends with an "S" then it is removed. If the return type is not a structure then a blank string is returned.

Usage

<PARAMETER_STRUCTURE_NOPLURAL>

Possible values

• String (structure name).

Alternate forms

- <parameter_structure_noplural>
- <Parameter_Structure_Noplural>
- <Parameter_structure_noplural>
- <ParameterStructureNoplural>
- $\bullet \quad <\! parameter Structure Noplural \! > \\$

Example of use

See also

- <PARAMETER_STRUCTURE>
- <PARAMETER_STRUCTURE_PLURAL>
- Other parameter loop expansion tokens

PARAMETER_STRUCTURE_PLURAL

Inserts the name of the STRUCTURE that defines the current parameters type. If the name of the structure does not end with an "S" then one is added. If the return type is not a structure then a blank string is returned.

Usage

<PARAMETER_STRUCTURE_PLURAL>

Possible values

• String (structure name).

Alternate forms

- <parameter_structure_plural>
- <Parameter Structure Plural>
- <Parameter_structure_plural>
- <ParameterStructurePlural>
- <parameterStructurePlural>

Example of use

See also

- <PARAMETER_STRUCTURE>
- <PARAMETER_STRUCTURE_NOPLURAL>
- Other parameter loop expansion tokens

PARAMETER_TYPE

Inserts the type of the current parameter.

Usage

<PARAMETER_TYPE>

Possible values

- Alpha
- BinaryHandle
- Decimal
- ImpliedDecimal
- Enumeration
- Handle
- Integer
- String
- Structure

Alternate forms

None

Example of use

See also

• Other parameter loop expansion tokens

Parameter Loop Expression Tokens

CodeGen includes parameter loop expressions are expressions that can only be used within the context of a parameter loop. CodeGen supports parameter loop expressions via the **IF expression** and **IF [expression]** tags. Note that loop utility expressions can also be used within parameter loops.

Expression	Description
<if alpha=""></if>	The parameter type is ALPHA.
<if binary_handle=""></if>	The parameter type is BINARY HANDLE.
<if collection=""></if>	The parameter exposes a collection.
<if collection_array=""></if>	The parameter exposes a collection via an array.
<if collection_handle=""></if>	The parameter exposes a collection via a dynamic memory handle.
<if collection_arraylist=""></if>	The parameter exposes a collection via an ArrayList.
<if coerce=""></if>	The parameter type is coerced to a different type.
<if corece_boolean=""></if>	The parameter type is coerced to type BOOLEAN.
<if corece_byte=""></if>	The parameter type is coerced to type BYTE.
<if corece_datetime=""></if>	The parameter type is coerced to type DATETIME.
<if< td=""><td></td></if<>	
CORECE_DATETIME_NULLABLE>	The parameter type is coerced to type NULLABLE DATETIME.
<if corece_decimal=""></if>	The parameter type is coerced to type DECIMAL.
<if< td=""><td>The management time is accounted to</td></if<>	The management time is accounted to
CORECE_DECIMAL_NULLABLE>	The parameter type is coerced to type NULLABLE DECIMAL.

<if corece_double=""></if>	The parameter type is coerced to type DOUBLE.
<if corece_float=""></if>	The parameter type is coerced to type FLOAT.
<if corece_int=""></if>	The parameter type is coerced to type INT.
<if corece_long=""></if>	The parameter type is coerced to type LONG.
<if corece_sbyte=""></if>	The parameter type is coerced to type SBYTE.
<if corece_short=""></if>	The parameter type is coerced to type SHORT.
<if corece_uint=""></if>	The parameter type is coerced to type UINT.
<if corece_ulong=""></if>	The parameter type is coerced to type ULONG.
<if corece_ushort=""></if>	The parameter type is coerced to type USHORT.
<if comment=""></if>	The method has an associated comment.
<if datatable=""></if>	The parameter is exposed as a DataTable.
<if date=""></if>	The parameter is a DATE.
<if date_yyyymmdd=""></if>	The parameter is a YYYYMMDD date.
<if date_yymmdd=""></if>	The parameter is a YYMMDD date.
<if date_yyjjj=""></if>	The parameter is a YYJJJ date.
<if date_yyyyjjj=""></if>	The parameter is a YYYYJJJ date.
<pre><if date_yyyymmddhhmmss=""></if></pre>	The parameter is a YYYYMMDDHHMMSS date/time.
<if decimal=""></if>	The parameter type is DECIMAL.

<IF DESCRIPTOR> The parameter is passed by descriptor. <IF ENUM> The parameter type is ENUM. The enumeration parameter being processed refers to an enumeration <IF FIRST_INSTANCE_OF_ENUM> that has not previously been encountered in this loop. The structure parameter being processed refers to a structure that <IF FIRST_INSTANCE_OF_STRUCTURE> has not previously been encountered in this loop. <IF HANDLE> The parameter type is HANDLE. <IF IMPLIED> The parameter type is IMPLIED DECIMAL. <IF IN> The parameter is an IN parameter. <IF INOUT> The parameter is an INOUT parameter. <IF IN OR INOUT> The parameter is an IN or INOUT parameter. <IF INTEGER> The parameter type is INTEGER. <IF OPTIONAL> The parameter is OPTIONAL. The parameter is an OUT <IF OUT> parameter. The parameter is an OUT or <IF OUT OR INOUT> INOUT parameter. The parameter is passed by <IF REFERENCE> REFERENCE. <IF REQUIRED> The parameter is REQUIRED. <IF STRING> The parameter type is STRING.

<IF STRUCTURE> The parameter type is

STRUCTURE.

<!F TIME> The parameter is a TIME.

<IF TIME_HHMMSS>
The parameter is an HHMMSS

time.

<IF VALUE> The parameter is passed by

VALUE.

Copyright © 2018 Synergex International, Inc.

Custom Tokens

CodeGen provides several mechanisms for performing custom processing, including:

- User defined expansion tokens
- User defined expression tokens
- Custom field loop expressions
- Custom data type mappings

Copyright © 2018 Synergex International, Inc.

User-Defined Expansion Tokens

User-defined expansion tokens are special tokens where a developer can determine both the name of the token as well as the value to be inserted when the token is encountered during code generation.

CodeGen supports several mechanisms to allow you to implement user-defined tokens, via:

- Environment variables.
- Include files.
- Command line tokens.
- Under defined token files.

If your template file depends on the developer providing a value for a particular user-defined token, you can tell CodeGen about that dependency. For more information refer to declaring user-defined token dependencies.

Environment Variables

As documented in generic tokens you can use the <ENV:variable> token in a template file and CodeGen will insert the value obtained from the environment variable into the output stream.

Include Files

Also documented in generic tokens you can use the <FILE:name.ext> token in a template file and CodeGen will read data from an external file and insert that data into the output stream.

Command Line Tokens

A quick way to declare used-defined tokens is to specify both a name and value on the command line using the -ut command line option, for example:

codegen —s <structure> -t <template> -ut MODEL_NS=MyApp.Model You can specify multiple user-defined tokens on the command line, space separated after the -ut option, for example:

```
codegen —s <structure> -t <template> -ut MODEL_NS=MyApp.Model VIEW_NS=MyApp.View
```

If your user-defined token value (the part after the equal character) contains double quotes then Windows will strip the quotes before the value is passed to CodeGen. In that case you must escape the double quote characters with a back-slash character, like this:

```
-ut MYMESSAGE=\"Hello World\"
```

By default, the replacement value user-defined tokens that are defined on the command line may not contain other tokens. If you wish to enable that functionality then you must tell CodeGen to treat user-defined tokens as preprocessor tokens via the -utpp command line option.

User-Defined Token Files

If you need to define several user-defined tokens then it is sometimes convenient to define them in a user-defined token file and name the user-defined token file via the -u command line option.

An example of a user-defined token file is:

```
; User defined tokens for Synergex Inc.
; 
<SUPPORT_PHONE>(800) 366-3472</SUPPORT_PHONE>
<SUPPORT_EMAIL>support@synergex.com</SUPPORT_EMAIL>
<WEBSITE>http://www.synergex.com</WEBSITE>
```

To tell CodeGen to read user-defined tokens from a file you would use the -u command line option. For example:

```
codegen —s <structure> -t <template> -u MYTOKENS: <usertokenfile>
```

You could then refer to your user defined tokens in your template files. For example:

display(tt,"Please call support at <SUPPORT_PHONE>") display(tt," or send an email to <SUPPORT_EMAIL>.")

Declaring User-Defined Token Dependencies

If CodeGen encounters something in a template file which is formatted like a token (i.e. delimited by < and > characters) but is not recognized as a token, then CodeGen will ignore it, and that text will remain in the output file that is created. This is necessary in order for CodeGen to be able to generate code for languages or dialects such as HTML, XML and XAML.

However, when it comes to user-defined tokens this can be a problem, because if a template contains something that the template developer intended to be a user-defined token which gets replaced, but someone who is subsequently generating code from the template neglects to specify the source of the user-defined tokens, the tokens will remain in the generated file and no error will be reported.

To address this situation template developers can use the <REQUIRES_USERTOKEN> token to declare that their template requires a particular user-defined token to be declared, either in a token file (-u) or on the command line (-ut).

If a templates declares that a user-defined token is required, but that user-defined token has not been declared when generating code, CodeGen will display an error and will stop processing the template.

Copyright © 2018 Synergex International, Inc.

User-Defined Expression Tokens

CodeGen supports user-defined expression tokens in all loops via the **<IF USERTOKEN**_*tokenname*> and

<IF NOT_USERTOKEN_tokenname> tags. User defined expression tokens
allow you to conditionally process template code based on the presence or
absence of a particular user defined token.

Multi-Line User Defined Token Expressions

Multi-line user token expressions occur when the opening and closing conditional tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
<****_LOOP>
[code]

<IF expression>conditional code
[<ELSE>
conditional code]
</IF[ expression]>
[code]
</**** LOOP>
```

Multi-line user token expressions cause one or more lines of output code to be included or excluded, based on the evaluation of the expression.

In-Line User Defined Token Expressions

In-line user token expressions occur when the opening and closing conditional tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
<****_LOOP>
[code]
[code] <IF expression> conditional code </IF[ expression]> [code]
```

```
[code]
```

```
</****_LOOP>
```

In-line user token expressions cause part of an output line to be included or excluded based on the evaluation of the expression.

Note: Multi-line expressions may be nested within one another, and in-line expressions may be nested within multi-line expressions. However, in-line expressions may not be nested inside one another.

Example

</IF>

If generating code with a command similar to:

codegen -s CUSTOMER -t FileIo -ut MYTOKEN=value

The template file code could include code based on the presence of the MYTOKEN user defined token like this:

```
<IF USERTOKEN_MYTOKEN>
This code will be included if the user token is specified </IF>
Or
<IF NOT_USERTOKEN_MYTOKEN>
This code will be included if the user token is specified
```

Custom Field Loop Expressions

CodeGen supports custom conditional expressions in field loops via the **<IF CUSTOM_expression>** template_code **</IF>** and **<IF NOT_CUSTOM_expression>** template_code **</IF>** tags. Custom field loop expressions test the presence or absence of some piece of text in a fields user text or long description.

There are two types of custom field loop expressions.

Multi-Line Custom Field Loop Expressions

Multi-line custom field loop expressions occur when the opening and closing conditional tags appear on separate lines in a template file, and delimit one or more entire lines of template code, like this:

```
<FIELD_LOOP>[code]
<IF CUSTOM_expression>
conditional code
[<ELSE>
conditional code]
</IF[ CUSTOM_expression]>
[code]
</FIELD_LOOP>
```

Multi-line custom field loop expressions cause one or more lines of output code to be included or excluded, based on the evaluation of the expression.

In-Line Custom Field Loop Expressions

In-line custom field loop expressions occur when the opening and closing conditional tags appear on the same line in a template file, and delimit part of a line of template code, like this:

```
<FIELD_LOOP>
[code]
[code] <IF CUSTOM_expression> conditional code </IF[
CUSTOM_expression]> [code]
[code]
```

```
</FIELD_LOOP>
```

In-line custom field loop expressions cause part of an output line to be included or excluded based on the evaluation of the expression.

Nested Expressions

Multi-line expressions may be nested within one another, and in-line expressions may be nested within multi-line expressions. However, in-line expressions may not be nested inside one another.

Custom Field Loop Expression Example

An example of using a custom field loop expression might be if you wanted to arbitrarily exclude certain fields in a structure from being processed within a field loop. You could decide to add the text IGNORE_ME to the fields user text, or to the fields long description. You could then test the presence, or absence of this string with custom expression tags, like this:

```
<FIELD_LOOP>
<IF CUSTOM_IGNORE_ME>
code for fields WITH the custom text
</IF>
<IF CUSTOM_NOT_IGNORE_ME>
code for fields WITHOUT the custom text
</IF>
</FIELD_LOOP>
```

Custom Data Type Mappings

CodeGen includes several field loop tokens that emit, based on the Synergy data type of a field, an appropriate data type for use in various other languages. These tokens are <FIELD_CSTYPE>, <FIELD_OCTYPE>, <FIELD_SNTYPE>, <FIELD_SQLTYPE> and <FIELD_VBTYPE>.

The data type mappings used by these tokens are hard coded to suitable default values within CodeGen. These default values generally follow the same rules that xfNetLink has always used when converting traditional Synergy types into .NET and Java types, and should work well in most scenarios. However if the defaults don't work in a particular scenario then it is possible to customize the values by specifying the data type mappings that you wish to use in an XML file.

The XML file must be defined in a specific way, and an example file called DataMappingsExample.xml is provided when you install or build CodeGen.

Within the XML file there are several main sections that correspond to the various languages that CodeGen supports:

Language	Field Loop Token	XML File Section
C#	<field_cstype></field_cstype>	CSharp
Objective C	<field_octype></field_octype>	ObjectiveC
Synergy .NET	<pre><field_sntype></field_sntype></pre>	SynergyDotNet
SQL	<field_sqltype></field_sqltype>	SQL
Visual Basic	<field_vbtype></field_vbtype>	VisualBasic

Within each of these main sections there are data type mappings for each of the field types that CodeGen supports. For example, the default mappings for C# look like this:

```
<CSharp> <Alpha>string</Alpha> <AlphaBinary>string</AlphaBinary>
```

```
<UserAlpha>string</UserAlpha>
 <UserNumeric>string</UserNumeric>
 <UserDate>string</UserDate>
 <UserTimeStamp>string</UserTimeStamp>
 <DateYYYYMMDD>DateTime</DateYYYYMMDD>
 <DateYYMMDD>DateTime</DateYYMMDD>
 <DateYYYYJJJ>DateTime</DateYYYYJJJ>
 <DateYYJJJ>DateTime</DateYYJJJ>
 <DateYYYYPP>string</DateYYYYPP>
 <DateYYPP>string</DateYYPP>
 <NullableDateYYYYMMDD>DateTime?</NullableDateYYYYMMDD>
 <NullableDateYYMMDD>DateTime?</NullableDateYYMMDD>
 <NullableDateYYYYJJJ>DateTime?</NullableDateYYYYJJJ>
 <NullableDateYYJJJ>DateTime?</NullableDateYYJJJ>
 <TimeHHMMSS>DateTime</TimeHHMMSS>
 <TimeHHMM>DateTime</TimeHHMM>
 <NullableTimeHHMMSS>DateTime?</NullableTimeHHMMSS>
 <NullableTimeHHMM>DateTime?</NullableTimeHHMM>
 <ImpliedDecimal>decimal</ImpliedDecimal>
 <SmallDecimal>int</SmallDecimal>
 <LargeDecimal>long</LargeDecimal>
 <Integer1>int</Integer1>
 <Integer2>int</Integer2>
 <Integer4>int</Integer4>
 <Integer8>long</Integer8>
 <Boolean>bool</Boolean>
 <Enum>int</Enum>
 <Binary>char(l)</Binary>
 <StructField>String</StructField>
 <AutoSequence>long</AutoSequence>
<AutoTime>long</AutoTime>
</CSharp>
```

In some data mappings you may need to specify the length of a variable (e.g. the Alpha data type for SQL), or length and precision (e.g. the ImpliedDecimal data type for SQL). To support this requirement CodeGen will search for and replace the following specific patterns of characters within data mapping values defined in your XML file:

Pattern of Characters	Replaced With
(l)	The length if the field in parentheses. E.g. (10)
(l.p)	The length and precision of the field, separated by a period, in parentheses. E.g. (10.2)
(l,p)	The length and precision of the field, separated by a comma, in parentheses. E.g. (10,2)

You only need to specify the parts of the XML file that correspond to the specific data mappings that you wish to change. For example, if all you want to do is make the <FIELD_CSTYPE> token insert the TimeSpan data type for D8 and D6 date fields (YYYYMMDD and YYMMDD) then you would configure your data mappings file like this:

Once you have configured your custom data mappings file you tell CodeGen to use it by using the **-cdm** command line option, or if you are using the CodeGenerator API via the CodeGenTaskSet.DataMappingsFile property.

Extending CodeGen

CodeGen can be extended, to implement custom expansion an expression tokens, by writing custom code in Synergy .NET or any other .NET language. For additional information, refer to:

- Writing Custom Expansion Tokens
- Writing Custom Expression Tokens

Copyright @ 2018 Synergex International, Inc.

Writing Custom Expansion Tokens

CodeGen supports the ability for developers to define custom expansion tokens by writing plug-in modules to provide logic associated with those tokens. This plug-in mechanism is implemented in a way that does not require developers to edit the core CodeGen source files. This is important because it means that it will not impede the ability to download future source code updates to the core CodeGen environment.

Writing Custom Expansion Tokens

Custom expansion tokens are implemented as classes in a class library assembly. To implement custom tokens a developer creates a class library assembly containing one or more extension classes, and simply drops the library into the main CodeGen folder alongside the other CodeGen assemblies.

If you prefer to have your custom token extensions loaded from a different location then you can set the environment variable CODEGEN_EXTDIR to the location of the custom token extensions assembly.

When CodeGen loads it will check for any custom token assemblies, and if it finds any it will dynamically load them. In order to achieve this a naming convention is used. The name of any custom extensions assembly must begin with the word "custom". For example you might chose to create an assembly named CustomTokens.dll.

Each class that implements a custom expansion token must implement the interface CodeGen.Engine.IExpansionToken which can be accessed by adding a reference to the CodeGenEngine.dll assembly.

Source Code Example

The CodeGen source code package includes a sample project called CustomExtensionsExample which contains examples of implementing all of the various types of custom expansion tokens. This project is configured not to build when you build the main solution because it is intended to only be an example. Developers are encouraged to develop their custom token processors in a separate solution.

The code below shows an example of a custom field loop expansion token:

```
import Systemimport System.Collections.Generic
import CodeGen.Engine
import CodeGen.RepositoryAPI
namespace CustomExtensionsExample
 ;;To implement a custom expansion token you must provide a class that
 ;;implements the CodeGen.Engine.IExpansionToken interface. The class
 ;;MUST have a default constructor. By default classes have an
 ;;implicit default constructor, but if you need to explicitly define
 ;;a constructor, make sure you don't define any parameters.
 ;;This token can be used in field loops, like this:
              <FIELD LOOP>
              My custom token produced this output:
<CUSTOM FIELD LOOP TOKEN>
              </FIELD LOOP>
 public class CustomFieldLoopToken implements IExpansionToken
  public property TokenName, String
   method get
   proc
    mreturn "CUSTOM_FIELD_LOOP_TOKEN"
   endmethod
  endproperty
  public property Description, String
   method get
   proc
    mreturn "An example of a custom field loop token."
   endmethod
  endproperty
  public property Validity, TokenValidity
```

```
method get
   proc
    mreturn TokenValidity.FieldLoop
   endmethod
  endproperty
  public property TokenCase, TokenCaseMode
   method get
   proc
    mreturn TokenCaseMode.UppercaseOnly
   endmethod
  endproperty
  public method Expand, String
   tkn, @Token
   template, @FileNode
   loops, @IEnumerable<LoopNode>
   endparams
  proc
   lambda doExpand(str, field)
   begin
    ;TODO: Add any code you need here, and return the string
    ;that is to be used to replace the token.
    mreturn "This is the output from CUSTOM_FIELD_LOOP_TOKEN"
   end
   mreturn TokenExpander.ExpandFieldLoopToken(tkn, template, loops,
doExpand)
  endmethod
 endclass
endnamespace
```

Copyright © 2018 Synergex International, Inc.

Writing Custom Expression Tokens

CodeGen supports the ability for developers to define custom expression tokens by writing plug-in modules to provide the logic associated with those tokens. This plug-in mechanism is implemented in a way that does not require developers to edit the core CodeGen source files. This is important because it means that it will not impede the ability to download future source code updates to the core CodeGen environment.

Writing Custom Expression Tokens

Custom expression tokens are implemented as classes in a class library assembly. To implement custom expression tokens a developer creates a class library assembly containing one or more extension classes, and simply drops the library into the main CodeGen folder alongside the other CodeGen assemblies.

If you prefer to have your custom extensions loaded from a different location then you can set the environment variable CODEGEN_EXTDIR to the location of the custom token extensions assembly.

When CodeGen loads it will check for any custom token assemblies, and if it finds any it will dynamically load them. In order to achieve this a naming convention is used. The name of any custom extensions assembly must begin with the word "custom". For example you might chose to create an assembly named CustomTokens.dll.

Each class that implements a custom expression token must implement the interface CodeGen.Engine.IExpressionToken which can be accessed by adding a reference to the CodeGenEngine.dll assembly.

Source Code Example

The CodeGen source code package includes a sample project called CustomExtensionsExample which contains examples of implementing all of the various types of custom expression tokens. This project is configured not to build when you build the main solution, because it is intended to only be an example. Developers are encouraged to develop their custom expression processors in a separate solution.

The code below shows an example of a custom field loop expression token:

```
import Systemimport System.Collections.Generic
import CodeGen.Engine
import CodeGen.RepositoryAPI
namespace CustomExtensionsExample
 ;;To implement a custom expression you must build a class that implements
the
 ;;CodeGen.Engine.IExpressionToken interface. The class MUST have a
default constructor.
 ;;By default classes have an implicit default constructor, but if you need to
 ;;explicitly define a constructor, make sure you don't define any parameters.
 ;;You can use this expression in field loops, like this:
 ;; <FIELD LOOP>
 ;; If you see YES then the expression evaluated to true: <IF
CUSTOM FIELD LOOP EXPRESSION>YES</IF>
 ;; </FIELD_LOOP>
 ;;
 public class CustomFieldLoopExpression implements IExpressionToken
  public property TokenName, String
   method get
   proc
    mreturn "CUSTOM FIELD LOOP EXPRESSION"
   endmethod
  endproperty
  public property Description, String
   method get
   proc
    mreturn "An example of a custom field loop expression."
   endmethod
  endproperty
```

```
public property Validity, TokenValidity
   method get
   proc
    mreturn TokenValidity.FieldLoop
   endmethod
  endproperty
  public method Evaluate, Boolean
   tkn, @Token
   template, @FileNode
   loops, @IEnumerable<LoopNode>
   endparams
  proc
   lambda doEvaluate(str, field, index)
   begin
    ;TODO: Add code here to determine the result of the expression, and
return true or false
    mreturn true
   end
   mreturn ExpressionEvaluator.EvaluateFieldLoopExpression(tkn,
template, loops, doEvaluate)
  endmethod
 endclass
endnamespace
```

The CodeGen API

As an alternative to using the command-line interface a developer could integrate CodeGen more tightly into their development environment by writing custom tools or utilities to generate code, using the core CodeGen environment.

To make this possible CodeGen provides a .NET API that developers can code directly against in order to generate code. The main classes in the CodeGen API are:

- CodeGen.Engine.CodeGenTaskSet
- CodeGen.Engine.CodeGenTask
- CodeGen.Engine.CodeGenerator

CodeGen API Example

This is a very simple example of using the CodeGen API. This code segment is essentially the same as using the command line:

```
codegen -s CUSTOMER -t DataClass -r -v
```

Here's the sample code:

```
;;Create a new task setdata taskset = new CodeGenTaskSet() taskset.LoggingLevel = LoggingLevel.Verbose
```

;;Create a task and define what it shold do data task = new CodeGenTask() task.Structures.Add("CUSTOMER") task.Templates.Add("DataClass") task.ReplaceFiles = true

```
;;Add the task to the task set taskset.Tasks.Add(task)
```

;;Create a code generator and tell it about the task set

```
data generator = new CodeGenerator(taskset)
;;Generate the code
generator.GenerateCode()
;;Did it work?
if (taskset.Complete) then
    ;;Good to go
else
    ;;Something failed!
```

You won't see anything happening when you execute this code because the CodeGen API doesn't implement any UI. But if you wanted to see the messages that are generated as the task set it processed you could register an event handler method against either taskset.Messages.CollectionChanged or task.Messages.CollectionChanged and monitor / report on the messages as they are generated. For example if you were in a console application you could log messages to standard out like this:

```
lambda messageFromTaskSet(sender, e)
begin
    if (e.Action==NotifyCollectionChangedAction.Add)
    begin
        data message, String
        foreach message in e.NewItems
        if (message!=^null)
        Console.WriteLine(message)
    end
end

;;Listen for messages from the taskset as it processes
taskset.Messages.CollectionChanged += messageFromTaskSet
```

CodeGenTaskSet Class

Represents the data for and results of a code generation session comprising of one or more CodeGen tasks. A CodeGenTaskSet object is processed by the CodeGenerator class in order to generate code.

For an example of generating code with the CodeGen API look here.

Syntax

Class Name: public class CodeGenTaskSet

Namespace: CodeGen.Engine

Assembly: CodeGenEngineShared.dll (later will move into

CodeGenEngine.dll)

Constructors

Name Description

CodeGenTaskSet Initializes a new instance of the CodeGenTaskSet class.

Methods

Name Description

De-serializes a previously saved

LoadFromFile(String,CodeGenTaskSet) XML file back to a CodeGenTaskSet

object.

SaveToFile(String)

Serializes the current objects data to

an XML file.

Required Properties Set by Calling Application Before Processing

Name Type Description

Tasks @ObservableCollection<CodeGenTask>

Optional Properties Set by Calling Application Before

Processing

Name	Type	Description
ContinueAfterError	Boolean	Continue processing after errors. If multiple templates, files or tasks are being processed then this option allows code generation to continue even though one template/structure combination has generated an error. Setting this property is the equivalent of using the -c command line option.
DataMappingsFile	String	Specifies that custom data mappings are loaded from an XML file. Refer to Customizing Data Type Mappings in the docs for more info. Setting this property is the equivalent of using the - cdm command line option.
Description	String	A description of the task set. Not used by CodeGen, but it may be useful for consuming code to display task set descriptions in their UI.
EchoCommands	Boolean	Display the command line used to start CodeGen. Setting this property is the equivalent of using the -e command line option.
ListGeneratedFiles	Boolean	List Files. Lists the names of any files generated when not running in verbose or debug logging modes. Setting this property is the equivalent of using the - lf command line option.
LoggingLevel	LoggingLevel	Logging level. Setting this property is the equivalent of using using one of the -v, -d, or -debug command line option.
LogTokenizerResults	s Boolean	Causes CodeGenerator to create log files containing the internal results of the code tokenization and parsing phases. Setting this property is the equivalent of using the -dt command line option.

OutputFolder	String	Output directory. Defaults to the current directory, or the directory specified with the CODEGEN_OUTDIR environment variable. You can use a full or relative path, or a logical followed by a colon (e.g. SRC:). Setting this property is the equivalent of using the -o command line option. A value set here provides a default value to each task in the task set.
RepositoryMainFile	String	Repository main file to use for this task set. Providing a value here provides a default value for each task in the task set. Setting this property is the equivalent of setting the RPSMFIL environment variable.
RepositoryTextFile	String	Repository text file to use for this task set. Providing a value here provides a default value for each task in the task set. Setting this property is the equivalent of setting the RPSTFIL environment variable. Input file directory containing templates.
TemplateFolder	String	Defaults to the current directory, or the directory specified with the CODEGEN_TPLDIR environment variable. You can specify a full or relative path, or a logical name followed by a colon (e.g. TPL:). Setting this property is the equivalent of using the -i command line option. Setting a value here provides a default value for all tasks in the task set.

Properties Set by CodeGenerator During Processing

Name	Type	Description
Complete Boolean		Indicates that all tasks
	Poologn	within the task set
	Boolean	completed without errors
		or warnings.

The last error message LastErrorMessage String

that was reported for all tasks in the task set.

All messages that were

Messages @ObservableCollection<String>reported while processing

the task set.

Events Raised During Processing

Name **Description**

Raised by CodeGenerator just before it starts processing the Starting Task Set

task set.

 $\label{eq:conditional} TaskSetComplete \\ \frac{Raised\ by\ CodeGenerator\ just\ after\ it\ finishes\ processing}{the\ task\ set.}$

Copyright © 2018 Synergex International, Inc.

CodeGenTask Class

Represents a single code generation task within a task set.

For an example of generating code with the CodeGen API look here.

Syntax

Class Name: public class CodeGenTask

Namespace: CodeGen.Engine

Assembly: CodeGenEngineShared.dll (later will move into

CodeGenEngine.dll)

Constructors

Name Description

CodeGenTask Initializes a new instance of the CodeGenTask class. CodeGenTask(String, Initializes a new instance of the CodeGenTask class for

String) a specified template and structure.

Required Properties Set by Calling Application Before Processing

Name Type Description

Structures List<String> The names of one or more repository structures to be processed.

The names of one or more template files to be

Templates List < String > processed. Specify only the base file name, not

including the location or file extension.

Optional Properties Set by Calling Application Before Processing

Name Type Description

The alias names for one or more structures. The number

Aliases	List <string></string>	of aliases specified must be less than or equal to the number of structures specified via the Structures property. Setting this property is the equivalent of using the - a command line option. Always load default button definitions from DefaultButtons.xml even if processing a UI Toolkit input
AlwaysLoadDefaultButton	s Boolean	window definition which may provide button definitions. Setting this property is the equivalent of using the - ba command line option. The height of a character in pixels. This property can be used to override the default
CharacterHeight	int	character height which is 25 pixels. Setting this property is the equivalent of using the - ch command line option. The width of a character in pixels. This property can be
CharacterWidth	int	used to override the default character width which is 12 pixels. Setting this property is the equivalent of using the - cw command line option. A description of the task. Not used by CodeGen, but it may
Description	String	be useful for consuming code to display task descriptions in their UI. Field prefix. Setting this
FieldPrefix	String	property is the equivalent of using the -prefix command

line option.

File Overrides. Allows you to specify which repository file definition is used for each structure being processing. This option is useful if the structure that you are using is assigned to multiple files because by default CodeGen will use the first assigned file. The number of file overrides provided must be less than or equal to the number of structures specified via the Structures property. Setting this property is the equivalent of using the **-fo** command line option.

Don't prefix group fields with group name. Setting this property is the equivalent of using the **-g f** command line option.

Don't use repository group field prefix. Setting this property is the equivalent of using the **-g r** command line option.

Don't expand implicit groups to individual fields. Setting this property is the equivalent of using the **-g i** command line option.

Exclude fields which are "Excluded by ReportWriter". Setting this property is the equivalent of using the **-f r** command line option.

FileOverrides List<String>

GroupFieldNoGroupPrefix Boolean

GroupFieldNoRpsPrefix Boolean

GroupNoExpand Boolean

 $Honor Exclude Report Writer\ Boolean$

HonorExcludeToolkit	Boolean	Exclude fields which are "Excluded by Toolkit". Setting this property is the equivalent of using the -f t command line option. Exclude fields which are
HonorExcludeWeb	Boolean	"Excluded by Web". Setting this property is the equivalent of using the -f w command line option. Exclude fields which are
IgnoreExcludeLanguage	Boolean	"Excluded by Web". Setting this property is the equivalent of using the -f l command line option.
IgnoreScriptFieldAttributes	s Boolean	When processing input from a UI Toolkit Window script file this option prevents script file input window field attributes from being used to override repository field attributes. This allows you to generate code based on only the fields listed in an input window, but using only the repository field definitions. Setting this property is the equivalent of using the -wn command line
IncludeOverlayFields	Boolean	option. Include overlay fields. Setting this property is the equivalent of using the -f o command line option. Multiple Structures. Specifies that the structures that follow the -s command line option should all be available to
MultipleStructures	Boolean	templates being processed, at

		the same time. Setting this property is the equivalent of using the -ms command line option. Multi-write file if same file generated more than once.
MultiWriteFiles	Boolean	Setting this property is the equivalent of using the - mw command line option. Namespace to use when replacing the <namespace> token. Codegen will determine the</namespace>
Namespace	String	default namespace using the environment variable SYNDEFNS. Setting this property is the equivalent of using the -n command line option. Never load any buttons (from DefaultButtons.xml or from a
NeverLoadAnyButtons	Boolean	window script). Setting this property is the equivalent of using the -bn command line option. Never use the default buttons defined in
NeverLoadDefaultButtons	Boolean	DefaultButtons.xml. Setting this property is the equivalent of using the -bd command line option. Output directory. Defaults to the current directory, or the directory specified with the CODEGEN_OUTDIR environment variable. You
OutputFolder	String	can use a full or relative path, or a logical followed by a

		colon (e.g. SRC:). Setting this property is the equivalent of using the -o command line option.
PrimaryKeyNumber	int	Overrides the key that is used within primary key blocks. Setting this property is the equivalent of using the - opk command line option.
ReplaceFiles	Boolean	Replace existing files. By default existing files are not replaced. Setting this property is the equivalent of using the - r command line option.
RepositoryFile	String	Repository file name to process. This option causes all structures that are assigned to the file definition to be available for use when processing templates. The result is the same as specifying multiple structures via the Structures property and setting the MultipleStructures property to true. Setting this property is the equivalent of using the -
RepositoryMainFile	String	file command line option. Repository main file to use for this task. The repository files can also be set on the CodeGenTaskSet to specify the same repository for all tasks. Setting this property is the equivalent of setting the RPSMFIL environment variable. Repository text file to use for

RepositoryTextFile	String	this task. The repository files can also be set on the CodeGenTaskSet to specify the same repository for all tasks. Setting this property is the equivalent of setting the RPSTFIL environment variable.
SelectionWindowScript	String	Window script selection list processing. When this option is used CodeGen will examine any fields which have selection windows specified and will attempt to populate the field selections property based on the selection items defined in the named selection window. Setting this property is the equivalent of using the - ws command line option. Subset of fields to use. Specify subsets in field user text strings with
Subset	String	@SUB=name; syntax. Setting this property is the equivalent of using the -subset command line option.
SubsetFields	List <tuple<string string="">></tuple<string>	Create a subset from the list

TemplateFolder String can specify a full or relative path, or a logical name followed by a colon (e.g. TPL:). Setting this property is the equivalent of using the i command line option. User defined token file. Specify a full or relative file spec of the file that contains user defined token definitions. UserTokenFile String The file spec can include Synergy logical names. Setting this property is the equivalent of using the **u** command line option. User defined tokens. Setting this property is the equivalent UserTokens List<UserToken> of using the -ut command line option. Process input from a UI Toolkit Window script file. By default the fields and buttons collection will be based on the first input window found in the script file, but this can be overridden using the #n WindowScript String qualifier, which specifies either a 1-based input window number or the name of the input window to process. Setting this property is the equivalent of using the -

Properties Set by CodeGenerator During Processing

w command line option.

Name Type Description

Complete	Boolean	Indicates whether the task completed without any
		errors and warnings.
Errors	int	Number of errors reported
LITOIS	1111	while processing the task.
		Number of files that failed
FilesFailed	int	to generate while processing
		the task.
		Names of files that were
FilesGenerated	ObservableCollection <string></string>	
		the task.
		Names of additional files
FilesProvided	ObservableCollection <string></string>	-
		processing the task.
T . T . 16		The last error message that
LastErrorMessag	e String	was reported while
		processing the task.
3.6		All messages that were
Messages	ObservableCollection <string></string>	
		the task.
T 47	• ,	Number of warnings
Warnings	int	reported while processing
		the task.

Events Raised During Processing

Name	Description
StartingTack	Raised by CodeGenerator just before it starts processing the
StartingTask	task.
TackComplete	Raised by CodeGenerator just after it finishes processing the task.
TaskComplete	task.

CodeGenerator Class

The CodeGenerator class is used to initiate code generation for a CodeGenTaskSet object. An application can cause code to be generated by creating an instance of this class, passing in a CodeGenTaskSet object, and then calling the GenerateCode method.

For an example of generating code with the CodeGen API look here.

Syntax

Class Name: public sealed class CodeGenerator

Namespace: CodeGen.Engine

Assembly: CodeGenEngine.dll

Constructors

Name Description

Initializes a new instance of the

CodeGenerator(CodeGenTaskSet)CodeGenerator class to process a

CodeGenTaskSet.

Methods

Name	Description
GenerateCode	Initiates code generation for the CodeGenTaskSet provided.
LoadExtensions	Loads custom extensions. This method is only exposed so that it may be used by unit tests in the CodeGen development environment, and should not be used by other applications.
Load Proforma Extensions	Loads BusinessCraft Proforma extensions. This method is only exposed so that it may be used by unit tests in the CodeGen development environment, and should not be used by other applications.

LoadSymphonyExtensions Loads Symphony Framweork extensions



UserToken Class

Represents a user-defined token.

For an example of generating code with the CodeGen API look here.

Syntax

Class Name: public class UserToken

Namespace: CodeGen.Engine

Assembly: CodeGenEngineShared.dll (later will move into

CodeGenEngine.dll)

Constructors

Name Description

UserToken Initializes a new instance of the UserToken class.

UserToken(String, Initializes a new instance of the UserToken class for a String) specified token name / token value pair.

Properties

Name Type Description

Name String The name the user defined token.

Value String The replacement value of the user defined token.

Example of Use

data favoriteColorToken, @UserToken, new UserToken("FAVORITE_COLOR","Blue")

data task, @CodeGenTask, new CodeGenTask()

task. User Tokens. Add (favorite Color Token)

LoggingLevel Enumeration

Defines available logging levels.

For an example of generating code with the CodeGen API look here.

Syntax

Class Name: public enum LoggingLevel

Namespace: CodeGen.Engine

Assembly: CodeGenEngineShared.dll (later will move into

CodeGenEngine.dll)

Definition

public enum LoggingLevel Normal, Verbose, Debug, DeveloperDebug endenum

Utility Programs

In addition to the main CodeGen program, the CodeGen distribution also includes several other utility programs:

Utility Description

CreateFile The CreateFile utility allows you to create ISAM files based on repository file or structure definitions.

MapPrep The MapPrep utility helps you to prepare a new repository structure for use with CodeGen's structure mapping capabilities.

RpsInfo The RpsInfo utility allows you to quickly display information from your repository.

Template Browser utility allows you to browse (and in some cases edit) CodeGen template files.

CreateFile Utility

The CreateFile utility allows you to create ISAM files based on repository file or structure definitions.

The command line options for the CreateFile utility are as follows:

CreateFile -f <fname> | -s <sname> [-out filespec] [-r] [-h]

- -f <fname> The name of the repository file definition to use.
- -s <sname>

The name of the repository structure definition to use.

-out <filespec>

The name of the file to create, overriding the file spec found in the repository file definition.

- -r Replace existing files. The default is not to replace existing files.
- -? Display this usage information.

When creating a file based on a repository file definition, the key specifications will be determined by the keys defined in the first structure that is assigned to the file.

When creating a file based on a repository file structure, the file specifications will be determined by the first file that the structure is assigned to.

CreateFile assumes that the repository key definitions are defined in the same sequence as the keys that they refer to. It is assumed that the first defined key is the primary key, the second key (if any) is the first alternate key, and so on.

CreateFile currently only supports creating ISAM files. Relative and sequential files will not be created.

Examples of Use

C:\> createfile -f CUSTFIL

DAT:customer.ism -> File created.

C:\> createfile -s CUSTOMER

DAT:customer.ism -> File created.

C:\> createfile -f CUSTFIL -f DAT:customer_save.ism

DAT:customer_save.ism -> File created.

MapPrep Utility

The MapPrep utility helps you to prepare a new repository structure for use with CodeGen's structure mapping capabilities. The you specify the name of an existing structure and the utility exports a repository schema for that structure, then post-processes the schema appending _MAPPED to the name of the structure, and adding all of the required structure to structure and field to field mapping relationships. The schema for the new mapped structure can then be loaded into the repository.

The -p command line option causes MapPrep to prompt you to enter new names for the structure and all fields. You can type ENTER to accept a default new structure name, or to keep the existing name for any field. This can save a lot of time as one of the primary reasons for using structure mapping is to allow you to generate code with more meaningful field names when the actual field names in the original structure are cryptic or un-descriptive.

For additional information refer to Structure Mapping.

Command Line Options

MapPrep -s structure [...] | * [-n newstr] [-p[s|f]] [-u] [-k] [-v] [-h]

- -s structure [...] | * Structure(s) to process, * to process all
- -n newstr

Name of new mapped structure (only when processing a single structure)

- -p Prompt for alternate structure and field names
- -ps Prompt for alternate structure name only
- -pf Prompt for alternate field names only
- -u Add mapping data to the user defined text field instead of the long description field.
 - -l Load new schemas directly into repository
- -ls Leave schema file after repository load

- -k Retain key definitions in mapped structure
- -v Verbose messages.
- -h Display this help / usage information

Examples of Use

MapPrep -s CUSMAS

The command above will export a repository schema for the existing structure CUSMAS, and will then create a new schema file called CUSMAS_MAPPED.SCH that describes a new structure called CUSMAS_MAPPED which contains the fields from the original CUSMAS structure, except any fields that were marked "excluded by language". The new schema file will contain special mapping codes from the new structure to the original structure, and from the remaining fields to the corresponding fields in the original structure. These data mapping codes will be added to a new blank line in the structure and fields long descriptions.

MapPrep -s CUSMAS -p

The command above will behave similar to the command described above, except that you will be prompted to enter new names for the mapped structure and its fields. The name of the schema file that is created will also be based on the structure name that you enter.

MapPrep -s CUSMAS -n CUSTOMER -pf

The command above will behave similar to the command described above, except that the name for the new structure (and schema file) have been provided on the command line. You will still be prompted to enter new names for all of the new structures fields.

RpsInfo Utility

The RpsInfo utility allows you to quickly display information from your repository.

The command line options for the RpsInfo utility are as follows:

RpsInfo [options] Displays information from repository

Options:

-s [structure [,...]] Display structure details

-v Display verbose information

-f Include field details

-k Include key details

-ks Include key segment details

-r Include relation details

-m Include format details

-t Include tag details

-f [file [,...]] Display file information

-k Include key information

-ks Include key segment information

Template Browser Utility

The Template Browser utility allows you to browse (and in some cases edit) CodeGen template files.

You can start the utility from a Start Menu shortcut, and you will see three tabs containing different types of templates.

- CodeGen Sample Templates (if installed)
- Symphony Framework Sample Templates (if installed)
- Your own templates.

A menu option allows you to specify the location of your own template files.

Symphony Framework Components

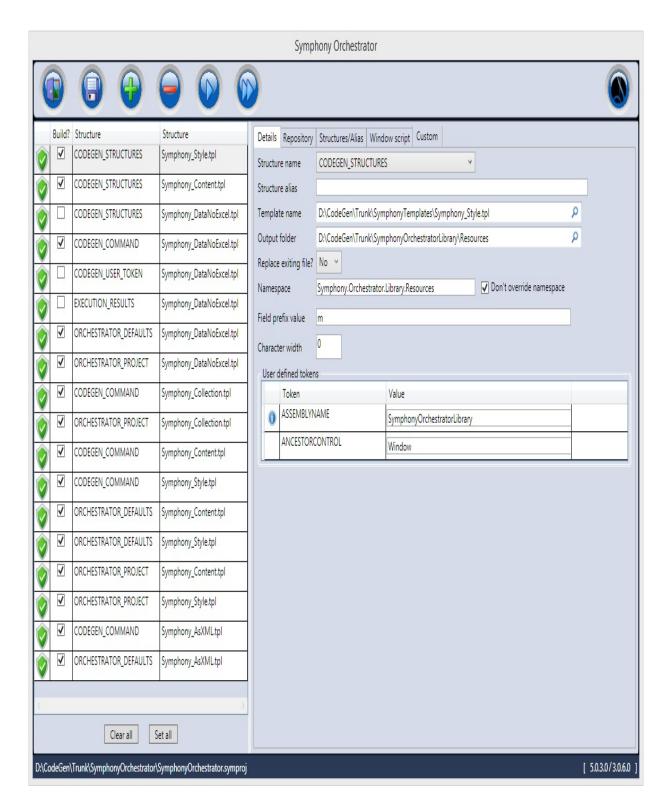
Several elements of the Symphony Framework are included with the CodeGen distribution, including:

- Symphony Orchestrator
- Symphony CodeGen Extensions

Symphony Orchestrator

Symphony Orchestrator

Symphony Orchestrator is a graphical Windows application that can be used to specify and execute code generation tasks.



Documentation for Symphony Orchestrator will be added here as soon as time permits.

Symphony CodeGen Extensions

Symphony CodeGen Extensions

Symphony CodeGen extensions are a collection of custom CodeGen custom extensions that have been developed by the Symphony Framework development team specifically for use when generating code that utilizes Symphony Framework.

- Custom Generic Expansion Tokens
- Custom Field Loop Tokens
- Custom Field Selection Loop Tokens

Symphony Generic Expansion Tokens

Symphony Framework - Generic Expansion Tokens

<SMPHONY_CLEARCONTENTLIST>

<SYMPHONY_CONVERTER_XML_NS>

<SYMPHONY_CONVERTER_RESOURCES>

<SYMPHONY_LOOPSTART>

Symphony Field Loop Expansion Tokens

Symphony Framework - Field Loop Expansion Tokens

- <SYMPHONY_ALPHA_SIZE>
- <SYMPHONY_CONTENTLISTFROMFIELD>
- <SYMPHONY_CONTENTLISTFROMWINDOW>
- <SYMPHONY_CONVERTER>
- <SYMPHONY_CONVERTER_PARAMETER>
- <SYMPHONY_DECIMAL_FORMAT>
- <SYMPHONY_DECIMAL_PLACES>
- <SYMPHONY_DEFAULT_VISIBILITY>
- <SYMPHONY_FIELD_PIXEL_WIDTH>
- <SYMPHONY_ITEMSSOURCE_PATH>

<symphony loopincrement=""></symphony>
--

<SYMPHONY_LOOPVALUE>

<SYMPHONY_SELWND_LENGTH>

<SYMPHONY_SEARCHBOX_COMMAND>

<SYMPHONY_UPDATE_TRIGGER>

Symphony Field Selection Loop Expansion Tokens

Symphony Framework - Field Selection Loop Expansion Tokens

<SYMPHONY_SELECTION_DESCRIPTION>

<SYMPHONY_SELECTION_VALUE>