SELVENTA™
Analyze. Accelerate. Clarify.

BEL Framework V1.2 Compiler/Assembler User Guide

Table of Contents

Introduction

This user guide will help you understand the use and concepts underlying Selventa's BEL Framework V1 Compiler/Assembler application. The BEL Framework Compiler/Assembler (BELC) is one component of Selventa's BEL Framework and is designed to compile scientific knowledge formatted as BEL statements into a computable network of biological facts and supporting evidence that can subsequently be used in knowledge-aware applications for reasoning and inference.

Other components of the BEL Framework include:

- BEL Workbench
- BEL Framework Tools
- BEL Framework API
- BEL Framework Web API

These components are documented separately.

Version Changes

Version 1.2 of the BEL Framework V1 Compiler/Assembler includes changes to the structure of the KAMs assembled in Phase IV. KAMs assembled with Version 1.2 of the compiler are not backwardly compatible with earlier versions. Additionally, Version 1.2 KAMs include the following changes:

- More exhaustive expansion of BEL Terms in Phase I resulting in increased densification of KAM nodes.
- Coupling of Molecular Activities between protein abundances and abundances of protein families.
- Memory usage and speed improvements.

System Requirements

The BELC application is programmed in Java 1.6 and requires a Java runtime environment of 1.6 or better to run. The compiler has been tested and verified on the following systems:

- Windows XP/SP2, 2gb RAM
- Windows 7/SP1, 4gb RAM
- OS/X 10.6, 2gb RAM
- Linux (various), 4gb RAM

Memory Requirements

The above system configurations should be used as guidelines however actual memory requirements will vary depending on the size of the largest BEL Document being compiled and the options selected during the augmentation phase (Phase III) of the compiler/assembler process.

Disk Space Requirements

The BEL Framework installation requires a small amount of disk space (< 50 mb). When running however, BELC will create many intermediate files and database tables and can require access to a much larger amount of disk space. The actual disk space requirements will vary depending on the number of BEL Documents being compiled and the number of statements in each BEL Document. The local disk space requirements are less if BELC uses remote databases for storing BEL Documents and/or compiled KAMs.

A rule of thumb for estimating the disk space requirements is to ensure that 10 times the size of the input BEL Documents is available.

Processing Overview

The BELC application can compile one or more BEL Documents into a single Knowledge Assembly Model (KAM). This section describes the processing of BEL Documents as they are assembled into a KAM.

Inputs

BELC takes one or more BEL Documents as input. The BEL Documents can be identified in one of 3 ways:

- As individual files on a diskdrive or file server
- As files in a folder on a diskdrive or file server
- As BEL Documents associated with a Document Group in a BEL Document Store

BELC can compile BEL Documents stored in XBEL or BEL Script format

- See the BEL Workbench documentation for details on managing BEL Document Groups.
- See the BEL Framework documentation for details on the XBEL and BEL Script formats.

Outputs

BELC assembles knowledge into a Knowledge Assembly Model (KAM). KAMs are stored in a KAM Store and are accessible by the BEL Framework API.

Processing

The BELC application is a multi-phase compiler/assembler. The current version of BELC implements 4 phases.

Phase I

Phase I compiles each BEL Document presented to the compiler to an intermediate version of a biological network called a *proto-network*.

This phase performs the following actions for each BEL Document passed into the compiler:

- Validates the structure of the BEL Document
- Verifies the syntax of each BEL Statement in the BEL Document
- Validates BEL Term arguments against designated namespace value domains
- Verifies the semantics of each BEL Term for each BEL Statement
- Expands *hasMembers* and *hasComponents* relationships into multiple *hasMember* and *hasComponent* relationships respectively
- Augments proto-networks by expanding nested BEL Terms to extract inner expressions by isolating inner abundance terms of activities and product and reactant components of reaction terms.
- Augments proto-networks by expanding nested BEL Statements and coupling BEL Terms by as adding *hasProduct* and *reactantIn* relationships to reactions and

connecting modified protein abundances to the unmodified protein abundances using *hasModification* and *hasVariant* terms.

- Creates a proto-network

Syntactic and semantic warnings and errors are output from Phase I. Each edge in the proto networks are supported by exactly one BEL Statement in a BEL Document and each node is supported by exactly one BEL Term.

The output of Phase I is a set of intermediate proto-networks stored in the compiler work area which are used as input to Phase II.

Phase II
Phase II takes as input the set of proto-networks from Phase I and merges them into a composite network. The merging process requires performing equivalencing of BEL Term arguments across namespaces where applicable.

This phase performs the following actions for each proto-network generated in Phase I:

- Equivalences each BEL Term argument across applicable namespaces
- Equivalences edges where source and object terms are equivalent, preserving relationship type.
- Merges the proto-network into a composite network by coalescing nodes and edges from the proto-networks where the BEL elements are equivalent.

The output of Phase II is a composite network where the nodes and edges from the proto-networks have been coalesced. The resulting network is saved in the compiler work-area Each node in the composite network is supported by one or more BEL Terms from one or more a BEL Documents and each edge is supported by one or more BEL Statements in one or more BEL Documents.

Phase III
Phase III takes as input the composite network from Phase II and augments the network by analyzing additional BEL Documents provided by the BEL Framework and injecting additional BEL Statements from these documents in order to make the resulting network more coherent and standardized for decision making.

This phase augments the composite network in three ways:

- Coupling Protein Family members to protein families
- Coupling Named Complex components to named complexes
- Coupling molecular activities between protein and protein family nodes
- Completing Gene activation pathways

Each of these sub-phases processes a BEL Document containing additional BEL Statements, selects the statements from the BEL Documents that are needed for the augmentation, and performs Phase I and Phase II on the selected statements to merge them into the composite network.

The output of Phase III is an augmented version of the Phase II composite network that has had additional nodes and edges injected into the network. The additional nodes and edges are supported by BEL Statements in one of the BEL Documents provide by the BEL Framework.

The augmentations performed in Phase III can be omitted by specifying the relevant command-line parameters. For more information on modifying Phase III behavior, see the Phase III Specific Arguments section on page 6.

Phase IV

Phase IV is the final assembly phase of BELC. This phase takes as input a composite network that has been generated by Phase III and exports the network as a KAM. The KAM is exported to a KAM Store where it can be accessed by the BEL Framework API.

This phase performs the following actions:

- Generates a KAM from a Phase III network and stores it in the KAM Store
- Registers the newly generated KAM in the KAM Catalog

The output of Phase IV is a ready-to-use KAM in the KAM Store.

Errors and Warnings

As the compiler/assembler works, it performs many types of tests and check to ensure the validity of the output. The majority of these tests are performed in Phase I where the document level and BEL Statement level syntactic checks, and the semantic checks for BEL Terms are performed.

Warnings

The majority of the checks performed by BELC will generate warnings if a check fails. Warnings are informational and are output to the console. By default, BELC will continue to the next phase if a warning is generated.

The majority of warnings will be generated from the semantic checking engine within BELC. A typical warning will look like the following:

```
SEMANTIC WARNING
       reason: term failed semantic checks
       signature: gtpBoundActivity(F:proteinAbundance) gtpBoundActivity
       function signatures: 1
       mismatched return types for signature
gtpBoundActivity(F:abundance) abundance
```

When a warning is generated, BELC will attempt to continue processing by ignoring the effect caused by the failed check. In the case above, the node created by BELC in the KAM will be generated but the return type for the *gtpBoundActivity()* function is incorrectly defined and might cause problems as the KAM is being used be applications.

Errors

BELC generates errors when a problem occurs which would cause the compiler to stop working correctly or where a significant effect on the resulting KAM would be generated such that the KAM might become unusable.

Errors can be generated by any phase of BELC however most errors will be generated in Phase I where documents are being checked for syntactic compatibility with the BEL Framework specification.

If an error is generated, the current phase will attempt to continue but the compiler will terminate at the end of the phase.

Pedantic Mode

BELC can optionally operate in pedantic mode where warnings are treated as errors and will terminate the compiler. This feature is useful for ensuring that complete compatibility with the BEL Framework specification is maintained and that resulting KAMs are fully functional.

See the command line arguments section for enabling pedantic mode.

Running the Compiler/Assembler

The BELC application is a command line application. The application can be run from a command window or shell from the installation directory for the BEL Framework.

BELC Command Line Arguments

This section identifies the command line arguments for BELC.

General Arguments

The following arguments are passed to the BELC compiler. Commonly used arguments have both a short and a long form.

Argument	Description
-h --help	Shows command line options.
-f [filename] --file	File name of an XBEL or BEL Script file to compile.
-p [path] --path	Path to a folder containing one or more XBEL or BEL Script files to compile.
-g [docgroup name] --document-group-name	Name of a Document Group in the BEL Document Store that contains one or more BEL Documents to compile.
-k [kam] --kam-name	Name of the KAM to create. If a KAM with the same name exists it will be overwritten if the --no-preserve option is set.
-d [description] --kam-description	A description for the KAM that will be created. Put the description in quotes. It may be necessary to escape the quotes (i.e. \”) for the full description to be recognized by BELC.
-s [filename] --system-config-file	Allows the user to specify the path to an alternative system configuration file to use to specify the BEL Framework configuration to use. The default is to use the default systems configuration file found in [installdir]/config.
--debug	Enables debug mode for the compiler/assembler. Debug mode preserves intermediate files and exports an XGMML version of the KAM for debugging purposes. The default is to disable debugging.
-v --verbose	Enables verbose mode. This mode outputs information about the different stages of each compiler/assembler phase. Usually used in conjunction with --timing to gather information about the compilation process. The default is to disable verbose mode.
-t --timing	Enables timing mode. This mode outputs times in seconds for each reported phase of the compiler/assemble. Usually used in conjunction with --verbose. The default is to disable timing mode.
--pedantic	Enables pedantic mode. This mode treats warnings as

	errors and will terminate the compiler/assembler at the end of a phase if a warning is generated in the phase. The default is to disable pedantic mode.
--no-preserve	If present, if a KAM with the same name exists it will be overwritten. The default is to preserve the KAM and exit with an error if the existing KAM will be overwritten.

Phase I Specific Arguments

These optional arguments are used to control the compilation of proto-networks during Phase I.

Argument	Description
--no-nested-statements	Modifies Phase I to force the compiler not to create a relationship between the subject term of a statement with a nested statement as the object, to the object term of the nested statement. The default is to couple nested statements.
--no-semantic-check	Bypasses the semantic checker. When set, this option will omit checking for semantic errors in the input documents. The default is to enable semantic checking.
--no-syntax-check	Bypasses the syntax checker. When set, this option will omit checking for syntax errors in the input documents. This is not recommended as it can lead to unpredictable KAM topologies. The default is to enable syntax checking.

Phase III Specific Arguments

These optional arguments are used to control the augmentation of the composite network during Phase III.

Argument	Description
--no-phaseIII	Executes Phase III in pass-through mode. This is the same as specifying --no-gene-scaffolding, --no-named-complexes and --no-protein-families.
--no-gene-scaffolding	Modifies Phase III to omit expanding the composite network to include gene activation pathways for gene products identified in a BEL Document input to the compiler. The default is to expand gene activation pathways for identified gene products.
--no-named-complexes	Modifies Phase III to omit coupling named complex members to existing nodes in the composite network. The default is to automatically create edges from named complexes in a BEL Document to their components.
--no-protein-families	Modifies Phase III to omit coupling protein family members to existing nodes into the composite network. The default is to automatically create edges from protein

	family nodes in a BEL Document to their members.
--expand-named-complexes	Modifies Phase III to inject named complex components and associated hasComponent edges when a component of a named complex is defined in a BEL Document. The default is not to inject named complex nodes and edges.
--expand-protein-families	Modifies Phase III to inject protein family nodes, members and associated hasMember edges when a member of a protein family is defined in a BEL Document. The default is not to inject protein family nodes and edges.

Additional Information

This section provides additional information that might be helpful to you.

Obtaining Technical Support

Technical support is available by phone or email during normal business hours (8am to 5pm EST).

Email Support
Send an email to support@belframework.org. Please make sure to include your name, a phone number where you can be reached, and details about the issue.

Phone Support
Please call Selventa's technical support line at (617) 851-5273 during normal support hours.

Learning More About Selventa's Software and Services

For all sales and other inquires, please contact:

Louis Latino
EVP Sales and Marketing
One Alewife Center,
Cambridge MA 02140

Phone: (617) 547-5421 x237
Email: llatino@selventa.com