# Resolute Function Libraries

While the analysis() capability in Resolute enables the execution of external plugins, a separate plugin is required for each analysis call. This is an inefficient mechanism for encapsulating a group of related functions. For example, a library of string manipulation functions (such as concat(), length(), substring(), etc.) would each require an individual plugin using the analysis() call, as in analysis(concat, str1, str2).

Resolute external function library support provides a mechanism for packaging multiple functions into a single plugin, which can then be called in a Resolute claim using the syntax `<LibraryName>.<LibraryFunction>(<Arg1>, <Arg2>, …)`. For example, the `concat` function in the StringLib string manipulation library is called as `StringLib.concat(str1, str2)`, and returns a string.

The following function libraries are included with Resolute:

- StringLib – String manipulations functions
- FileAccess – File attribute and content accessor functions
- ShellCmd – Command line environment functions for executing programs in a shell
- MySQL – SQL functions for database access

In addition, the BriefCASE tool includes additional Resolute function libraries that may be bundled with Resolute in the future:

- AgreeLib – Functions that provide access to AGREE contracts contained in an AADL model
- ModelAccess – Functions that provide additional information about an AADL model

# StringLib

StringLib is a Resolute Function Library for string manipulation. The library contains many of the basic functions found in the Java String class. The StringLib functions and their usage are described below. The syntax for calling a StringLib function in Resolute is `StringLib.<function_name>(<arg1>, <arg2>, …)`.

- `concat(s1 : string, s2 : string) : string`

  Returns the concatenation of string `s2` to the end of `s1`.

- `contains(s1 : string, s2 : string) : bool`

  Returns `true` only if string `s2` is contained within string `s1`. Returns `false` otherwise.

- `endsWith(s1 : string, s2 : string) : bool`

  Returns `true` only if string `s1` ends with string `s2`. Returns `false` otherwise.

- `hashCode(s : string) : int`

  Returns the hashcode for string `s`.

- `indexOf(s1 : string, s2 : string) : int`

  Returns the index within string `s1` of the first occurrence of string `s2`.

- `lastIndexOf(s1 : string, s2 : string) : int`

  Returns the index within string `s1` of the last occurrence of string `s2`.

- `stringLength(s : string) : int`

  Returns the length of string `s`. Note that this library method name differs from the name in the `java.lang` package. This is necessary to avoid conflict with the Resolute built-in `length()` function.

- `matches(s1 : string, s2 : string) : bool`

  Returns `true` if string `s1` matches the regular expression given in string `s2`. Returns `false` otherwise.

- `replace(s1 : string, s2 : string, s3, string) : string`

  Replaces each substring `s2` in string `s1` that matches the literal target sequence with the literal replacement sequence specified in string `s3`. The replacement proceeds from the beginning of the string to the end. The modified string is returned.

- `split(s1 : string, s2 : string) : [string]`

  Returns a list of strings that results from splitting string `s1` around matches of the given regular expression specified in string `s2`.

- `startsWith(s1 : string, s2 : string) : bool`

  Returns `true` only if string `s1` starts with the substring `s2`. Returns `false` otherwise.

- `substring(s : string, i1 : int, i2 : int) : string`

  Returns a string that is a substring of string `s`. The substring begins at index `i1` and extends to the character at index `i2 - 1` of string `s`. The length of the returned substring is therefore `i2 - i1`.

- `toLowerCase(s : string) : string`

  Converts all of the characters in string `s` to lower case and returns the string.

- `toUpperCase(s : string) : string`

  Converts all of the characters in string `s` to upper case and returns the string.

- `trim(s : string) : string`

  Returns string `s`, with any leading and trailing whitespace removed.

# FileAccess

FileAccess is a Resolute Function Library for accessing file system files and file attributes. The library contains many of the basic accessor functions found in the Java File class. No functions for modification of file or their attributes are included. The FileAccess functions and their usage are described below. The syntax for calling a FileAccess function in Resolute is `FileAccess.<function_name>(<arg1>, <arg2>, …)`. Most FileAccess functions take a file path as an argument. The file path can either be relative or absolute. If it is relative, it is assumed to be relative to the AADL project containing the system under evaluation.

- `canExecute(filePath : string) : bool`

  Returns whether the file given by `filePath` is executable.

- `canRead(filePath : string) : bool`

  Returns whether the file given by `filePath` has read privileges.

- `canWrite(filePath : string) : bool`

  Returns whether the file given by `filePath` has write privileges.

- `compareTo(filePath1 : string, filePath2 : string) : int`

  Compares the two path names given by `filePath1` and `filePath2`.

- `fileExists(filePath : string) : bool`

  Returns whether the file given by `filePath` exists in the file system.

- `getAbsolutePath(filePath : string) : string`

  Returns the absolute path of the file given by `filePath`.

- `getContents(filePath : string) : string`

  Returns the contents of the file given by `filePath`.

- `getParent(filePath : string) : string`

  Returns the parent directory of the file given by `filePath`.

- `getFreeSpace(filePath : string) : int`

  Returns the number of unallocated bytes of the partition given by `filePath`.

- `getName(filePath : string) : string`

  Returns the name (without the full path) of the file given by `filePath`.

- `getTotalSpace(filePath : string) : int`

  Returns the size (in bytes) of the partition given by `filePath`.

- `getUsableSpace(filePath : string) : int`

  Returns the usable space (in bytes) of the partition given by `filePath`.

- `hashcode(filePath : string) : int`

  Returns the hashcode corresponding to the pathname of `filePath`.

- `isAbsolute(filePath : string) : bool`

  Returns whether the file given by `filePath` is absolute.

- `isDirectory(path : string) : bool`

  Returns whether the path given by `path` is a directory.

- `isFile(path : string) : bool`

  Returns whether the path given by `path` is a file.

- `isHidden(filePath : string) : bool`

  Returns whether the file given by `filePath` is hidden.

- `lastModified(filePath : string) : int`

  Returns an integer representing the time (in milliseconds since the epoch) that the file given by `filePath` was last modified.

- `length(filePath : string) : int`

  Returns the length of the file given by `filePath`.

- `list(path : string) : [string]`

  Returns a list of files in the directory specified by `path`.

# ShellCmd

[To be completed]

# MySQL

[To be completed]

# AgreeLib

AgreeLib provides functions for accessing AGREE `assume`, `guarantee`, `lemma`, and `assert` statements in AADL component types and implementations. Note that this library defines the `agree_spec` data type for representing AGREE statements in Resolute. The AgreeLib functions and their usage are described below:

- `hasAgreeProperty(c : component, s : string) : bool`

  Returns `true` only if component `c` contains the AGREE statement with the ID specified by `s`. Returns `false` otherwise.

- `agreeProperty(c : component, s : string) : agree_spec`

  Returns the AGREE statement in component `c` with the ID specified by `s`. An exception will be thrown if the AGREE statement does not exist. To avoid the exception, call the `hasAgreeProperty()` function prior to calling `agreeProperty()`.

- `agreePropertyID(a : agree_spec) : string`

  Returns the ID of AGREE statement `a`. AGREE statements that do not contain an ID will return an empty string.

- `agreePropertyDescription(a : agree_spec) : string`

  Returns the text description of AGREE statement `a`.

- `agreeProperties(c : component) : {agree_spec}`

  Returns the set of AGREE statements contained in component `c`. If component `c` does not contain any AGREE statements, an empty set is returned.

- `agreeAssumes(c : component) : {agree_spec}`

  Returns the set of AGREE `assume` statements contained in component `c`. If component `c` does not contain any AGREE `assume` statements, an empty set is returned.

- `agreeGuarantees(c : component) : {agree_spec}`

  Returns the set of AGREE `guarantee` statements contained in component `c`. If component `c` does not contain any AGREE `guarantee` statements, an empty set is returned.

- `agreeLemmas(c : component) : {agree_spec}`

  Returns the set of AGREE `lemma` statements contained in component `c`. If component `c` does not contain any AGREE `lemma` statements, an empty set is returned.

- `agreeAsserts(c : component) : {agree_spec}`

  Returns the set of AGREE `assert` statements contained in component `c`. If component `c` does not contain any AGREE `assert` statements, an empty set is returned.

- `isAssume(a : agree_spec) : bool`

  Returns `true` only if AGREE statement `a` is an `assume` statement. Returns `false` otherwise.

- `isGuarantee(a : agree_spec) : bool`

  Returns `true` only if AGREE statement `a` is a `guarantee` statement. Returns `false` otherwise.

- `isLemma(a : agree_spec) : bool`

  Returns `true` only if AGREE statement `a` is a lemma statement. Returns `false` otherwise.

- `isAssert(a : agree_spec) : bool`

  Returns `true` only if AGREE statement `a` is an assert statement. Returns `false` otherwise.

# ModelAccess

[To be completed]