What’s new in ColdFusion 2025.0.07 Alpha refresh release

Select from additional LLM providers

Use models from multiple providers to align applications with organizational policies, latency considerations, and regional deployment constraints. A unified abstraction layer now supports OpenAI, Ollama, Anthropic, and Mistral, so you configure provider settings centrally instead of rewriting integration code.

Maintain context across AI conversations

Maintain durable conversational context so you design AI workflows that reuse prior prompts and responses instead of restarting each interaction. Persistent chat memory integrates with Redis, Memcached, and Ehcache, providing structured storage for chat state under your own deployment controls. There’s also support for in-memory chat memory.

Configure vector stores for RAG

Standardize retrieval-augmented generation pipelines so you tune distance metrics, dimensions, and indexing strategies consistently across environments. Centralized configuration now defines connection details and defaults for Milvus, Pinecone, Qdrant, and Chroma, including metricType, indexType, namespaces, and connection timeouts.

Adopt consistent AI service APIs

Standardize AI integrations so your code calls chat, document, and retrieval services through a consistent, compact API surface. Names including VectorStore, EmbeddingModel, chatModel, aiService, documentProcessor, rag, mcpserver, and mcpclient, together with member functions load, split, transform, transformSegments, and ingest, replace earlier method signatures.

Summary of changes

Old name	New name
GetVectorstoreClient	VectorStore
GetEmbeddingModel	EmbeddingModel
CreateMCPServer	MCPServer
CreateMCPClient	MCPClient
GetChatModel	ChatModel
GetAIService	AIService
GetDocumentProcessingService	DocumentProcessor
Rag	SimpleRAG
LoadDocuments	Load
SplitDocuments	Split
TransformDocuments	Transform
TransformTextSegments	TransformSegments
IngestSegments	Ingest
DocumentProcessor	DocumentService

New MCP functions

- ListMCPServers(): Returns a struct containing all currently registered MCP servers in the application. The keys are server names and values are the corresponding McpServer instances. Useful for monitoring and managing multiple MCP server instances.
- RemoveMCPServer(name,version): Removes the MCP server with the specified name from the registry. Returns true if the server was found and removed, false if no server with that name existed. Use this to clean up servers that are no longer needed.
- MCPServerExists(name,version): Checks whether an MCP server with the specified name is currently registered. Returns true if the server exists, false otherwise. Use this to verify server availability before attempting operations.

Vector Store parameter updates

Adobe ColdFusion standardizes the configuration requirements for Milvus, Pinecone, and Qdrant to maintain consistent behavior across supported vector store providers. These updates define mandatory fields, clarify default values, and specify connection and retry parameters for retrieval-augmented generation workflows.

Updated parameter requirements

Milvus

- **url**: mandatory (“localhost:19530”)
- **dimension**: mandatory
- **collectionName**: optional (“default”)
- **databaseName**: optional (“default”)
- **metricType**: optional (“COSINE”)
- **indexType**: optional (“HNSW”)
- **apiKey**: optional
- **username**: optional
- **password**: optional
- **connectionSettings**: optional
    - **callTimeout** (60s)
    - **connectionTimeout** (20s)
    - **keepAlive** (true)
    - **keepAliveTime** (30s)
    - **keepAliveTimeout** (5s)
    - **idleTimeout** (600s)
    - **deadline** (180s)
- **retrySettings** (optional)
    - **maxRetries** (10)
    - **retryOnRateLimit** (true)
    - **initialBackoff** (500ms)
    - **maxBackoff** (3s)
    - **backoffMultiplier** (3)

Qdrant

- **collectionName**: optional (“default”)
- **url**: mandatory
- **apiKey**: optional
- **dimension**: mandatory
- **metricType**: optional (“COSINE”)
- **connectionSettings**
    - **callTimeout** (60s)
    - **connectionTimeout** (20s)
    - **keepAlive** (true)

    - **keepAliveTime** (30s)
    - **keepAliveTimeout** (5s)
    - **idleTimeout** (600s)
- **retrySettings**
    - **maxRetries** (10)
    - **initialBackoff** (500ms)
    - **maxBackoff** (3000ms)
    - **backoffMultiplier** (3)

Pinecone

apiKey: mandatory
indexName: mandatory
namespace: optional, "default"
either one of podConfig or serverlessConfig is required and all the fields are necessary in their sections.
podConfig:
dimension:
environment:
podType:
serverlessConfig:
dimension: mandatory
cloud:
region:
deletionProtection: enabled/disabled

Updated language function behavior

The following table summarizes new and updated language-level functions, including purpose, parameters, return values, and behavioral details.

Function	Description	Syntax	Parameters	Returns / Behavior
FileMismatch	Compares two files byte by byte to identify differences.	`FileMismatch(String filePath1, String filePath2)`	**filePath1**: Path to the first file. **filePath2**: Path to the second file.	**–1** if files are identical; otherwise the 0-based byte position of

				the first mismatch.
FileReadLines	Reads multiple lines from a file starting at a specific position, with optional backward reading.	`FileReadLines(Object file, long startLine, long count)`	**file**: `FileStreamWrapper` or file path string. **startLine**: 0-based line number to start reading from. **count**: Optional line count; reads to end-of-file when omitted.	Returns the requested range of lines; supports negative *count* values to read backward from *startLine*.
ArrayGetAt / array.at	Accesses an array element with support for negative indexing.	`ArrayGetAt(List array, int index)` `array.at(index)`	**array**: Target array. **index**: 1-based positive index or negative index (for example, –1 for the last element).	Returns the element at the specified position; negative indices count from the end of the array.
StructGetEntries / struct.entries	Converts a struct to an array of key–value entry structs.	`StructGetEntries(Map struct)` `struct.entries()`	**struct**: Source struct.	Returns an array of structs, each with **key** and **value** properties.

DirectoryCreate	Creates directories with control over intermediate paths and existing directories; backward compatible with the previous DirectoryCreate behavior.	`DirectoryCreate( String path, boolean createPath, boolean ignoreExists)`	**path**: Directory path to create. **createPath**: Creates intermediate directories (default: `true`). **ignoreExists**: Skips creation when the directory already exists (default: `false`).	Creates the target directory according to the specified flags; retains compatibility with existing DirectoryCreate usage.
ArrayFindLast / **array.find	Finds the last occurrence of a value or the last element that matches a closure in an array.	`ArrayFindLast(List array, Object value_Callback)` `array.findLast(value)`	**array**: Target array. **value_Callback**: Value to match or closure for custom matching logic.	Returns the 1-based index of the last matching element, or 0 when no match is found.
ListFindLast	Searches a delimited string list from the end for a value.	`ListFindLast(String list, String value, String delimiter, boolean includeEmptyFields)`	**list**: Delimited string list. **value**: Value to search for. **delimiter**: List delimiter. **includeEmptyFields**: Matches	Returns the 1-based position of the last matching list element, or 0 when

			behavior of `ListFind()` for empty fields.	the value is not found.
QueryIsEmpty	Tests whether a query object contains any rows.	`QueryIsEmpty(query)`	**query**: Query object to inspect.	Returns `true` when the query has no rows; returns `false` when at least one row exists.
round	Rounds numeric values with optional control over decimal precision.	`round(number)` `round(number, decimals)`	**number**: Value to round. **decimals**: Optional number of decimal places.	Returns *number* rounded to the nearest integer or to the specified number of decimal places.
queryNew	Creates a query object with optional initial row data via argumentCollection.	`queryNew(columnList, rowData=...)` (rowData via ArgumentCollection)	**columnList**: Columns for the query. **rowData**: Optional initial row data passed using ArgumentCollection.	Returns a new query with defined columns and optional initial rows.

GetApplicationMetadata	Returns metadata for all valid application settings, including defaults.	`GetApplicationMetadata()`	*None.*	Returns all 43 valid application settings with proper default values, not only explicitly defined settings.
javaCast("int", ...)	Casts values to an integer with support for the full unsigned 32-bit range using two's complement wrapping.	`javaCast("int", value)`	**value**: Numeric value to cast.	Accepts values from –2,147,483,648 to 4,294,967,295; values between 0 and 4,294,967,295 are treated as unsigned 32-bit integers with two's complement wrapping.

Asynchronous workflow enhancements

New asynchronous functions

asyncAllOf

Name	Description
Function	`asyncAllOf(Array futures)`
Purpose	Waits for all futures in the input array to complete before returning.
Parameters	**futures** – Array of `Future` objects that should all complete.
Return / Behavior	Returns when **all** futures in the array have completed; propagates completion state according to the combined outcomes.
Usage notes	Use when downstream logic must wait for the full set of asynchronous operations before proceeding.

asyncAnyOf

Name	Description
Function	`asyncAnyOf(Array futures)`
Purpose	Returns as soon as the first future in the array completes.
Parameters	**futures** – Array of `Future` objects to monitor.
Return / Behavior	Completes when **any** future finishes (the fastest one), and reflects that future's completion state.
Usage notes	Use when only the first completed result is required, or when you want to race multiple asynchronous operations.

orTimeout

Name	Description
Function	`future.orTimeout(long timeoutMs)`
Purpose	Enforces a maximum wait time on a future and fails it when the timeout is exceeded.
Parameters	**timeoutMs** – Timeout duration in milliseconds.

Return / Behavior	If the future does not complete within *timeoutMs*, it fails with a `TimeoutException`.
Usage notes	Use to avoid unbounded waits when integrating with slow or unreliable asynchronous operations.

completeOnTimeout

Name	Description
Function	`future.completeOnTimeout(Object value, long timeoutMs)`
Purpose	Guarantees completion of a future by supplying a default value when the timeout expires.
Parameters	**value** – Default value to use on timeout. **timeoutMs** – Timeout duration in milliseconds.
Return / Behavior	If the future is not completed within *timeoutMs*, it completes with *value* instead of failing.
Usage notes	Use when a safe fallback value is acceptable and you want to prevent timeout exceptions.

coldfusion.async.emptyfuture.blocking (flag)

Name	Description
Flag	`coldfusion.async.emptyfuture.blocking`
Purpose	Controls whether `get()` on an incomplete `EmptyFuture` blocks or throws immediately.
Type	Boolean configuration flag.
Default	`true`
Behavior (true)	`get()` blocks until another thread or process completes the `EmptyFuture`, matching the new, standard async coordination behavior.
Behavior (false)	`get()` throws `TaskNotCompletedException` immediately for an incomplete `EmptyFuture`, preserving the previous behavior for backward compatibility.

Usage notes	Set to `false` only when existing applications depend on the earlier immediate-failure behavior.

ColdFusion Builder plugin for Visual Studio enhancements

Dictionary enhancements for AI workflows

- **MCP client and server creation:** Dictionary support added for defining and managing MCP clients and servers.
- **AI Service and Chat Model configuration:** Enables dictionary-based setup for AIService and ChatModel instances.
- **VectorStore and EmbeddingModel setup:** Dictionary entries now support VectorStore clients and embedding model creation.
- **RAG workflows:** Dictionary support extended to Simple RAG and Document Services workflows.

Project Manager workflow refinement

The plugin modifies the existing Project Manager workflow to allow project creation or import operations without requiring a workspace. It also supports opening previously created workspaces through **File > Open Workspace from File**.

Note: You can open previously created workspaces from the File menu. This might not work in some cases. As a workaround, create or import the project again.

Stability and quality improvements

The plugin also includes important bug fixes and general improvements to enhance reliability across supported development workflows.