BASIC IDE code auto-completion - LibreOffice

Enhancing Developer Productivity in LibreOffice BASIC

Personal Details

Name : Devansh Varshney
E-mail : varshney.devansh614@gmail.com
Country : India
IRC name : devansh
Phone number : +91-7906973823
Time Zone : UTC +5:30

Details of the task

Description - Improving the Code Auto-completion Feature within the LibreOffice BASIC IDE.

Goal - To develop a significantly enhanced and user-friendly code auto-completion system for the LibreOffice BASIC IDE. This will eliminate current limitations, improve developer productivity, and provide a coding experience comparable to modern IDEs. The key features to be implemented include: Dot Operator Completion, Context-Aware Completion (Ctrl+Space), Parameter Info, and an Object Browser.

Skills - C++, UNO, BASIC, Reading other's code

Size - ~350 Hrs

Difficulty - Hard

Mentor - Hossein Nourikhah (IRC: hossein | mail: hossein@libreoffice.org)

Blueprint

1. **Introduction: Enhancing LibreOffice BASIC Developer Productivity**
    - 1.1 The Problem: Current IDE Limitations
    - 1.2 The Goal: A Modern Code Assistance System
    - 1.3 Key Features Overview

2. **Project Goals & Justification (Addressing User Needs)**
    - 2.1 Goal: Comprehensive Object Browser (tdf#165785)
    - 2.2 Goal: Reliable Code Completion (tdf#165786, tdf#66185)
    - 2.3 Goal: Inline Code Intelligence (tdf#92253)
    - 2.4 Goal: Default Availability & Stability (tdf#165780)

3. **Proposed Solution Architecture**
    - 3.1 Component 1: The Master Analyzer (Information Engine)
    - 3.2 Component 2: The Knowledge Base & Cache
    - 3.3 Component 3: The Interactive Catalog (Object Browser UI)
    - 3.4 Component 4: The Suggestion Assistant (Completion Logic & UI)
    - 3.5 Component 5: The Quick Tip Provider (Info Tooltip Logic & UI)
    - 3.6 Component 6: Integration & Configuration Layer

4. **Codebase Mapping & Implementation Details**
    - 4.1 Overview
    - 4.2 Area 1: BASIC Core Engine (basic/source/) Modifications
    - 4.3 Area 2: BASIC IDE UI (basctl/) Modifications & Additions
    - 4.4 Effort Focus & Existing Code Leverage

5. **Project Timeline and Milestones**
    - 5.1 Overall Approach: Backend First, UI Follows
    - 5.2 Phase 1: Core Analysis Engine & Cache Implementation
    - 5.3 Phase 2: Object Browser and Core Completion UI
    - 5.4 Phase 3: Info Tips, Integration, Testing & Polish
    - 5.5 Phase 4: Final Submission & Wrap-up

6. **Conclusion: Value and Impact**
    - 6.1 A Foundational Improvement for BASIC Development
    - 6.2 Key Benefits Summarized
    - 6.3 Addressing Community Needs
    - 6.4 Future Potential

7. **About Me**

Color coding -

Color - File

Color - Class

Color - Directory

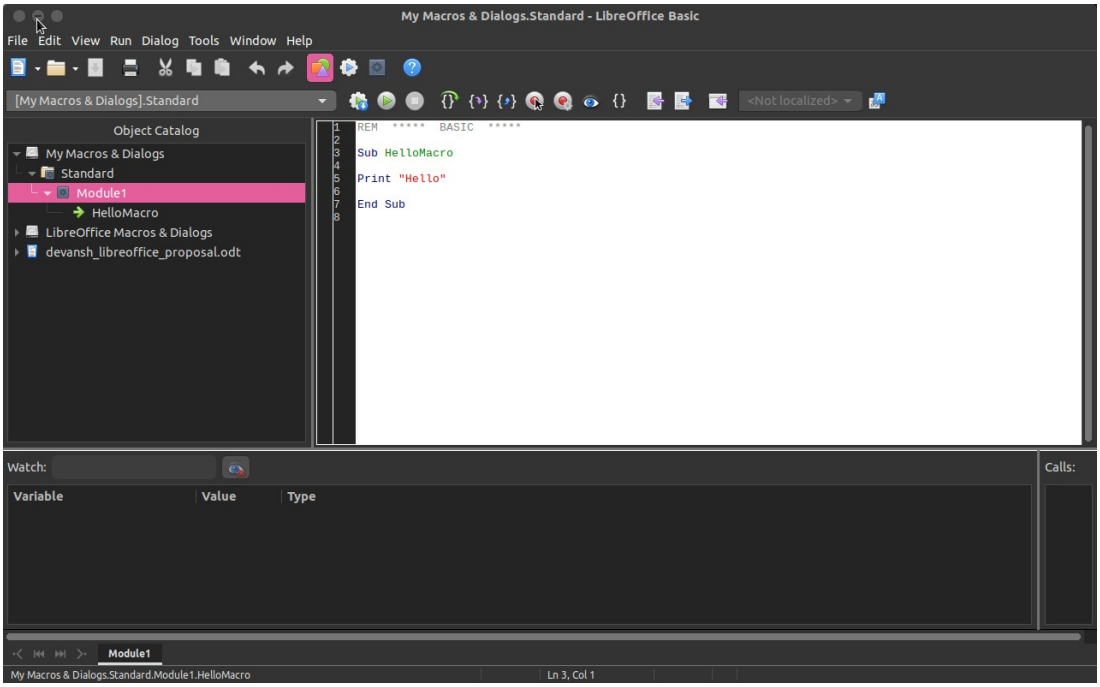
Color – Method/Heading

Introduction - Enhancing LibreOffice BASIC Developer Productivity

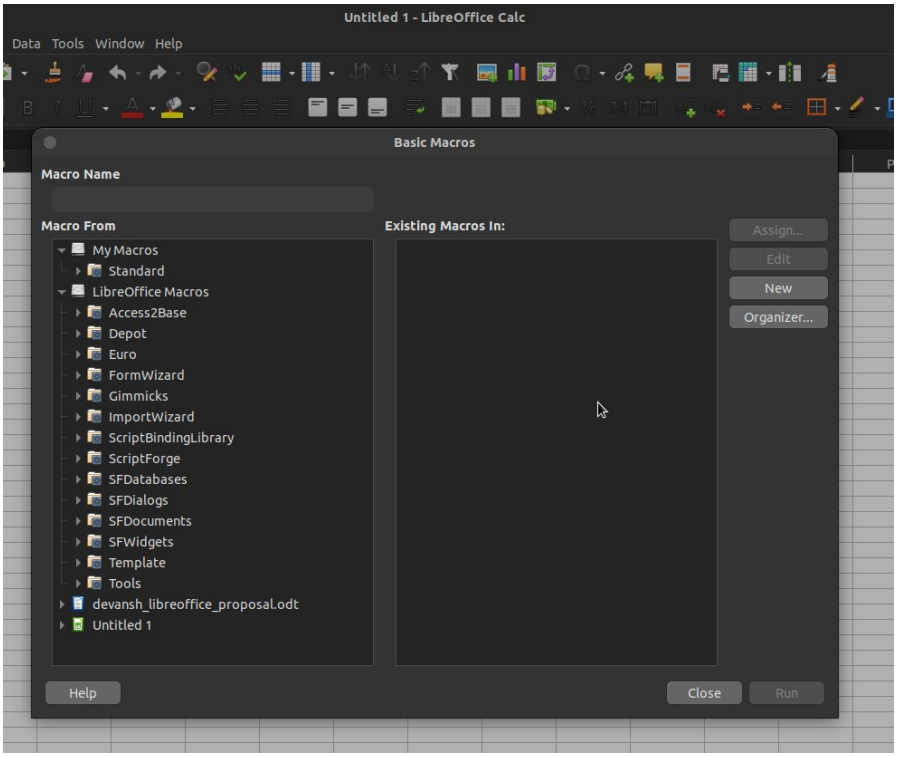
1.1 The Problem: Current IDE Limitations

The current LibreOffice BASIC Integrated Development Environment (IDE) provides essential tools for macro development, but its code assistance features, while present, are significantly limited and often fall short of user expectations in modern development environments. The official LibreOffice Help describes the existing features found under Tools > Options > LibreOffice > Basic IDE:

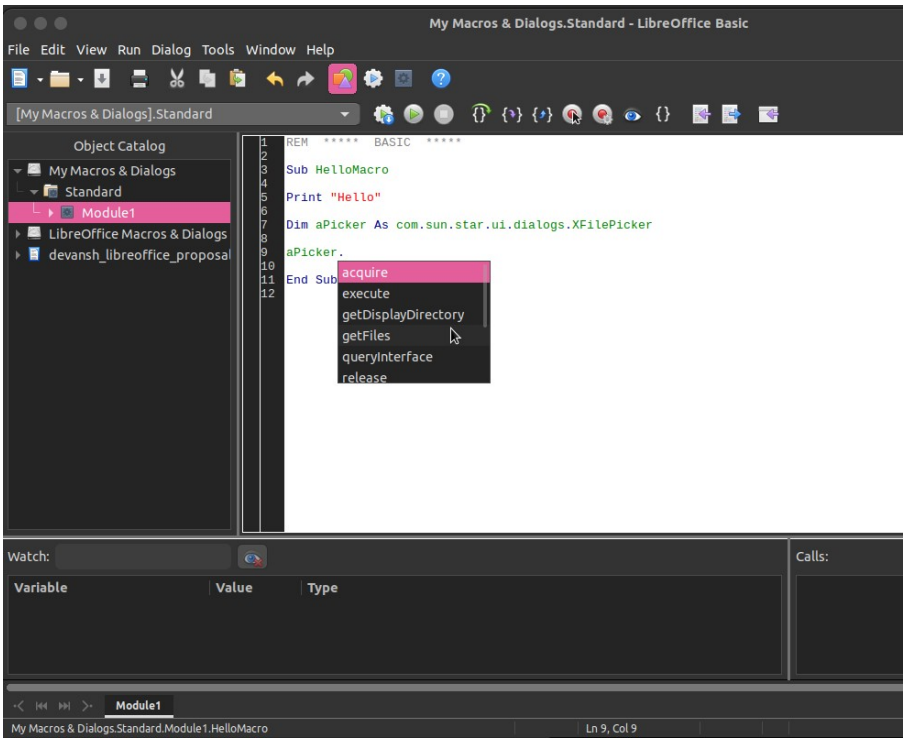
- **Existing "Code Completion":**
  - *Functionality:* The IDE offers a "dot completion" feature (Enable code completion setting). When typing a variable name followed by a period (.), a list box *may* appear showing potential methods.
  - *Major Limitation:* As documented, this primarily works only when the variable is explicitly declared using a UNO extended type (Dim aVar As com.sun.star...) and the Use extended types option is enabled. It explicitly **"does not work on a generic Object or Variant Basic types,"** which are very common in BASIC scripting. Furthermore, it often only lists methods, not properties.



Screenshot of Basic IDE = Tools > Macros > Organise Macros > Basic > New



Screenshot of Tools > Macros > Organise Macros > Basic Macros



Screenshot of current limited completion popup after typing 'aPicker.'

- **Existing "Code Suggestion" Helpers:** The IDE also provides basic typing aids:
    - *Autocorrection:* Corrects the case of variables and keywords.
    - *Autoclose quotes/parenthesis/procedures:* Adds matching closing characters.

      Note: While helpful, these are distinct from the intelligent code completion that is the focus of this project.
- **"Use extended types" Setting:** This language feature setting is currently a prerequisite for the limited dot completion to function at all.
- **Overall Issue:** The current system lacks context-aware suggestions (Ctrl+Space), offers no inline information (Quick Info/Parameter Info), doesn't provide a way to browse available elements (Object Browser), and its core completion feature has very narrow applicability. Its functionality is also confusingly tied to the global "Experimental Features" flag (tdf#165780), even if enabled in the BASIC IDE options.

These limitations hinder developer productivity, increase the likelihood of errors, and make discovering and utilizing the full potential of LibreOffice BASIC and the UNO API challenging, especially for less experienced users. User feedback, like the discussion in https://ask.libreoffice.org/t/in-lo-basic-ide-code-completion-seems-to-work-for-some-uno-objects-but-not-for-others/89863 , and https://ask.libreoffice.org/t/enable-code-completion-basic/14067/2 highlights this frustration.

1.2 The Goal: A Modern Code Assistance System

This project aims to address these shortcomings by developing a robust, modern code assistance system for the LibreOffice BASIC IDE. The objective is to provide developers with intelligent, context-aware support comparable to features found in other contemporary IDEs, making BASIC scripting significantly more efficient and user-friendly.

1.3 Key Features Overview

The core deliverables of this project are:

1. **Object Browser:** An integrated catalog displaying all accessible libraries, modules, classes, objects, methods, properties, and variables (tdf#165785).
2. **Enhanced Code Completion:**
    - Reliable "dot completion" for members of various object types (UNO, BASIC).
    - Context-aware "Ctrl+Space" completion for keywords, variables, procedures, etc. (tdf#165786, tdf#66185).
3. **Code Intelligence Tooltips:**
    - "Quick Info" displaying type/definition on hover.
    - "Parameter Info" showing function/subroutine signatures during calls (tdf#92253).
4. **Default Enablement:** Making these enhanced features stable and accessible to all users by default (tdf#165780).

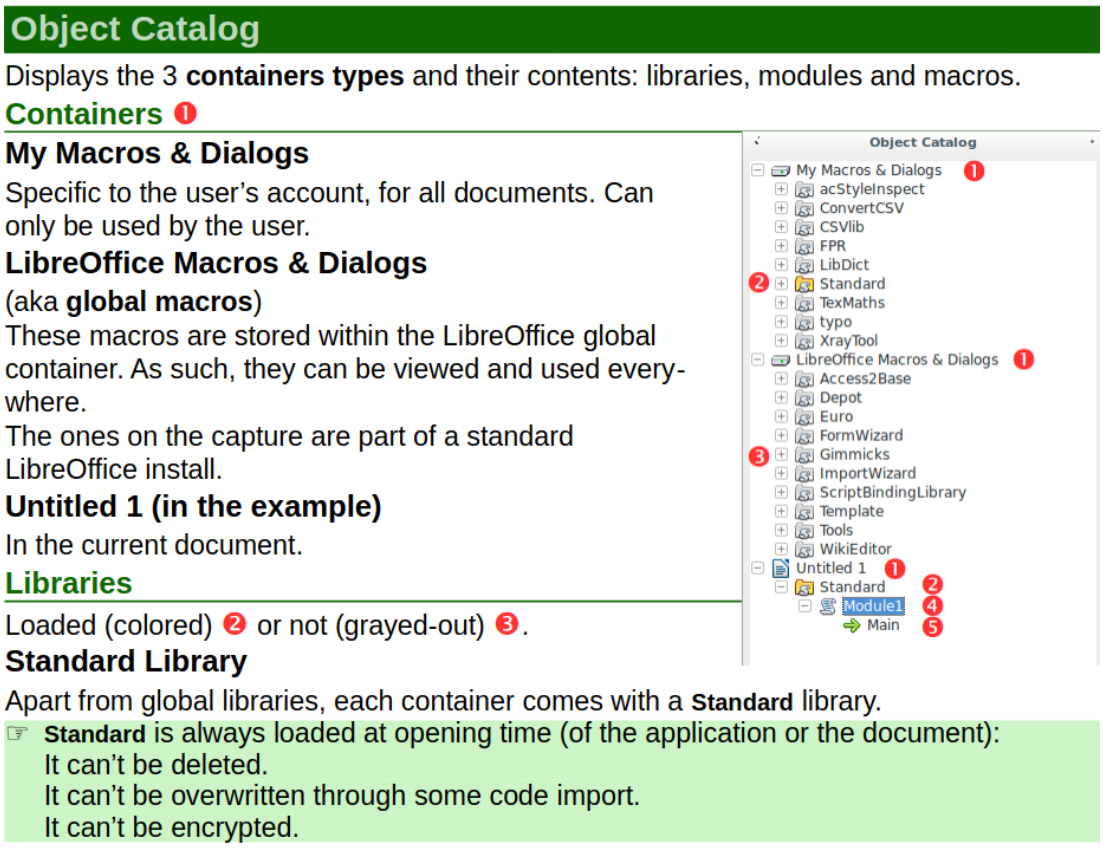
This proposal outlines the design, codebase integration strategy, and implementation plan to build this enhanced code assistance system, following a first-principles approach of understanding the core requirements and constructing the solution methodically.

Part 2: Project Goals & Justification (Addressing User Needs)

The goal of implementing a modern code assistance system is driven directly by specific, long-standing user requests and documented limitations within the current LibreOffice BASIC IDE. This section details these requirements, linking them to the relevant Bugzilla tickets.

2.1 Goal: Comprehensive Object Browser (tdf#165785)

- **Problem:** Developers currently lack an integrated tool to discover the available programming elements (globals, libraries, modules, classes, members like Subs, Functions, Properties). They rely on external documentation or code inspection, hindering exploration and learning. The existing "Object Catalog" sidebar is insufficient.



Screenshot of the current limited "Object Catalog" sidebar

- **Requirement (tdf#165785):** Implement an "Object Browser" within the IDE. This tool should allow users to:
    - Browse all available elements, categorized logically (e.g., <globals>, specific libraries).
    - View members (Properties, Subs, Functions, Constants) of selected elements (Modules, Classes, Objects).

    - See usage information (like function signatures) for selected members.
    - Include elements like Modules, Classes, Enums, Properties, Subs, Functions, and Constants.
    - Ideally, include search functionality.
- **Project Scope:** This project will build the backend data structures needed to populate such a browser and implement the user interface component itself within the IDE framework (basctl).

2.2 Goal: Reliable Code Completion (tdf#165786, tdf#66185)

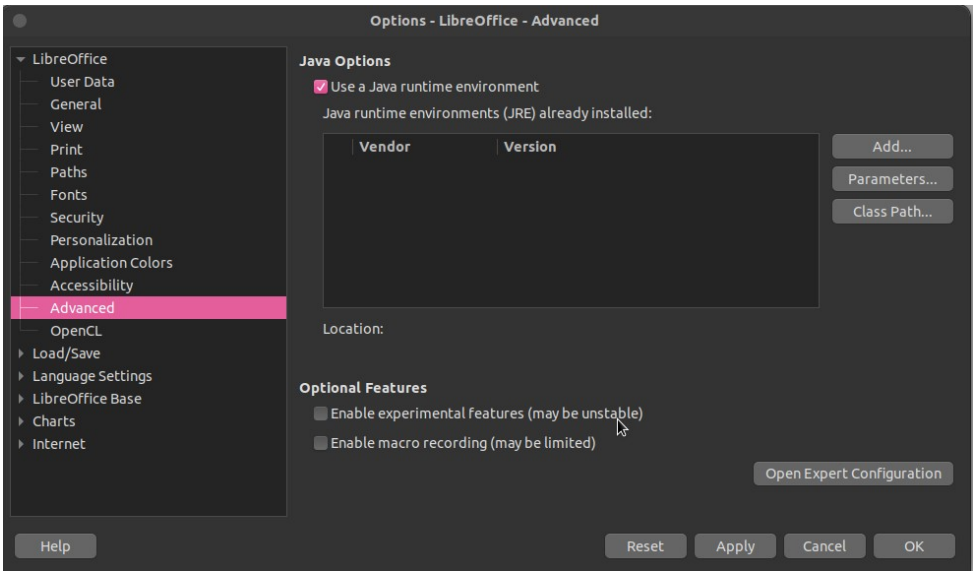
- **Problem:** The existing code completion is highly limited and unreliable.
    - *Dot Completion:* Works inconsistently, primarily only for certain UNO objects, and often fails to list all relevant members (tdf#66185). It doesn't support standard BASIC variables or user-defined types effectively.
    - *Contextual Completion:* There is no mechanism (like Ctrl+Space) to get suggestions for partially typed variable names, function names, or keywords based on the current scope and context.
- **Requirement (tdf#165786, tdf#66185):**
    - Implement context-aware completion triggered by Ctrl+Space, suggesting relevant keywords, variables, functions, objects, etc., based on the information gathered for the Object Browser.
    - Overhaul dot (.) completion to reliably show methods and properties for *all* relevant object types (UNO, BASIC objects, user types), not just a limited subset.
- **Project Scope:** This involves creating a robust code analysis engine to understand types and scope, a comprehensive cache, and integrating this backend with the IDE's editor (basctl/EditorWindow) to trigger and display suggestions via the existing (but adapted) CodeCompleteWindow.

2.3 Goal: Inline Code Intelligence (tdf#92253)

- **Problem:** Developers frequently need to look up the type of a variable or the parameters required by a function/subroutine, interrupting their workflow.
- **Requirement (tdf#92253):** Provide "IntelliSense"-like features:
    - *Quick Info:* Show a tooltip with the definition/type of the identifier under the cursor/mouse hover.
    - *Parameter Info:* Display a tooltip listing the parameters (name, type, order) of a function/subroutine when the opening parenthesis (is typed.
- **Project Scope:** This requires extending the backend cache to store detailed signature information and integrating tooltip display logic into the IDE's editor (basctl/EditorWindow).

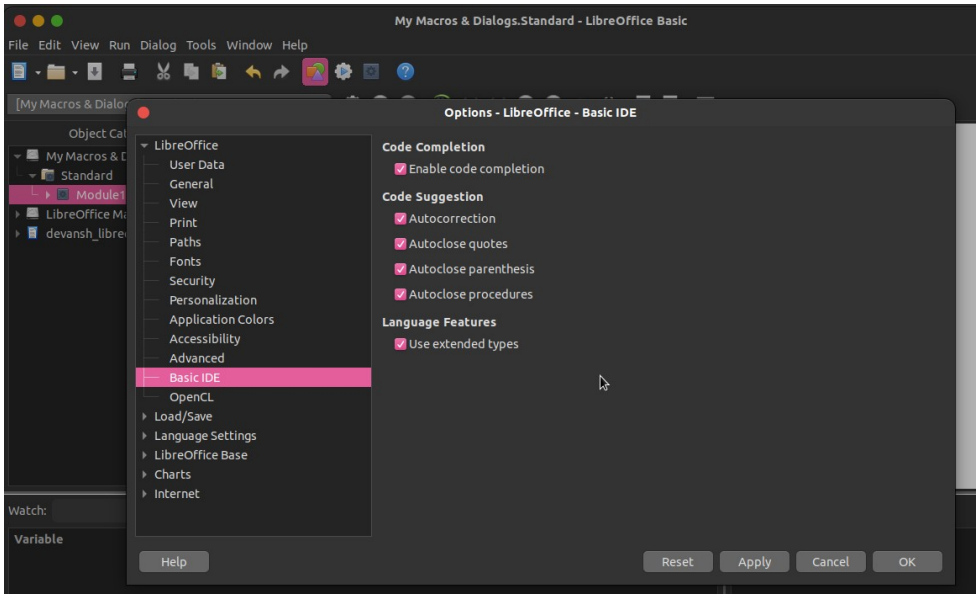
2.4 Goal: Default Availability & Stability (tdf#165780)

- **Problem:** Existing limited completion features are hidden behind the global "Experimental Features" flag, causing confusion and limiting accessibility (tdf#165780).



Go

(menu):Tools → Options → Advanced (place a tick in Enable experimental features + press OK)



Go

(menu):Tools → Options → BASIC IDE Options (all options are now available for selection)

- **Requirement:** Once the enhanced features are implemented and stable, they should be enabled by default, removing the dependency on the experimental flag.
- **Project Scope:** This involves ensuring the final implementation is robust and performant, followed by adjusting the relevant configuration settings (likely involving basic/source/classes/codecompletecache.cxx and potentially build flags).

Part 3: Proposed Solution Architecture

To achieve the goals outlined above, we propose an architecture composed of several interconnected components working within the existing LibreOffice BASIC IDE structure.

3.1 Component 1: The Master Analyzer (Information Engine)

- **Role:** Responsible for analyzing BASIC source code (both in the editor and from libraries) and introspecting UNO objects to gather comprehensive information about all available programming elements.
- **Functionality:**
    - **Incremental Parsing:** Analyze the code currently being edited in basctl/EditorWindow in near real-time. This analysis must be error-tolerant and capable of extracting symbol information (variables, types, scopes, procedure stubs) even from incomplete or syntactically incorrect code.
    - **Library Analysis:** Query the basic/BasicManager for loaded libraries (application and document-specific). For loaded BASIC libraries (basic/StarBASIC), parse their modules (basic/SbModule) to extract definitions of public Subs, Functions, Constants, User Defined Types (UDTs), and Globals, including full signatures (parameter names, types, ByRef/ByVal).
    - **UNO Introspection:** Utilize UNO reflection APIs (like XIntrospectionAccess potentially accessed via basic/SbUnoObject) to determine the methods, properties, and interfaces of UNO objects encountered (e.g., `Dim obj As com.sun.star.sheet.XSpreadsheet`).
    - **Scope Resolution:** Determine the scope and visibility (Public, Private, Global, Module, Procedure) of all identified symbols.
- **Implementation Notes:** This likely involves adapting the existing parsing infrastructure in basic/source/comp/ (SbiParser, SbiSymPool) to support incremental, on-demand analysis triggered by editor events. New logic will be needed to handle partial code and extract detailed signature information not strictly required by the compiler. Querying mechanisms for loaded libraries (StarBASIC, SbModule) might need refinement.

3.2 Component 2: The Knowledge Base & Cache

- **Role:** Acts as the central, optimized repository for the information gathered by the Master Analyzer. Provides fast query interfaces for the UI components.
- **Functionality:**
    - **Storage:** Store detailed information about symbols: name, type (BASIC native, UDT, UNO interface/service), scope, source location (library/module/line), full signatures for procedures, members of UDTs and classes, properties/methods of UNO types.
    - **Indexing:** Implement efficient indexing structures to allow quick lookups based on:
        - Partial symbol names (for completion).

        - Current scope (to filter suggestions).
        - Type names (for Object Browser and resolving variable types).
        - Context (e.g., finding members of a specific type for dot-completion).
    - **Cache Management:** Handle updates efficiently when the source code changes (triggering re-analysis by the Master Analyzer) or when libraries are loaded/unloaded. Needs mechanisms for invalidation and repopulation.
- **Implementation Notes & Cache Population:**
    - This component will replace the simplistic basic/CodeCompleteDataCache. It will likely require new classes, perhaps located in basic/source/classes/ or a dedicated subdirectory.
    - **Population Strategy:**
        - **Initial Scan:** Upon IDE startup or opening a document/library, the Master Analyzer performs an initial scan of the relevant loaded libraries (application libraries, document libraries) and the currently open module(s). It parses these sources and uses UNO reflection for known UNO types to populate the cache with baseline information (globals, library members, UDTs, etc.).
        - **Incremental Updates:** As the user types in the EditorWindow, editor modification events trigger the Master Analyzer to re-parse the *affected region* of the current module (e.g., the current procedure or block). The Analyzer extracts updated symbol information for that scope and sends updates to the cache.
        - **UNO Introspection (On-Demand/Cached):** Introspection of UNO objects can be relatively slow.
            - For known UNO types declared via Dim As com.sun.star..., introspection can happen during the initial scan or the first time the type is encountered, and the results (methods, properties) stored permanently in the cache associated with that type name.
            - For variables declared As Object or As Variant assigned a UNO object at runtime, dot-completion would still need runtime introspection *if* the type cannot be inferred statically, but this project focuses primarily on compile-time/IDE-time information. The goal is to maximize what can be determined *before* running the code.
    - **Performance:** Data structures like hash maps (for quick name lookup) and potentially prefix trees (tries?) or sorted lists (for partial name matching) will be crucial for achieving performance required for interactive features. Caching introspection results is key.
- **Goals Addressed:** Provides the fast data source for Object Browser, Code Completion, and Infotips.

3.3 Component 3: The Interactive Catalog (Object Browser UI)

- **Role:** The user interface for browsing the Knowledge Base.
- **Functionality:**
    - Display hierarchical view (Scopes -> Libraries -> Modules/Classes -> Members).
    - Show details (type, signature, etc.) for selected items.

    - Filter/Search capabilities.
- **Implementation Notes:** New UI component within basctl/. Will query the Knowledge Base & Cache. Needs integration with basctl/Shell.

3.4 Component 4: The Suggestion Assistant (Completion Logic & UI)

- **Role:** Provides inline code completion suggestions.
- **Functionality:**
    - Trigger on **.** and Ctrl+Space.
    - Analyze context around the trigger point.
    - Query Knowledge Base & Cache for relevant suggestions (keywords, variables, members based on type).
    - Display suggestions using an adapted basctl/CodeCompleteWindow.
- **Implementation Notes:** Major enhancements to basctl/EditorWindow. Relies heavily on fast queries to the Knowledge Base & Cache. Dot completion needs accurate type info from the Master Analyzer.

3.5 Component 5: The Quick Tip Provider (Info Tooltip Logic & UI)

- **Role:** Displays Quick Info and Parameter Info tooltips.
- **Functionality:**
    - Trigger on hover and (.
    - Identify code element under cursor/at parenthesis.
    - Query Knowledge Base & Cache for type/signature information.
    - Display information in a non-intrusive UI tooltip.
- **Implementation Notes:** New logic in basctl/EditorWindow. Requires UI element for tooltip display. Relies on detailed signature/type info in the Knowledge Base & Cache.

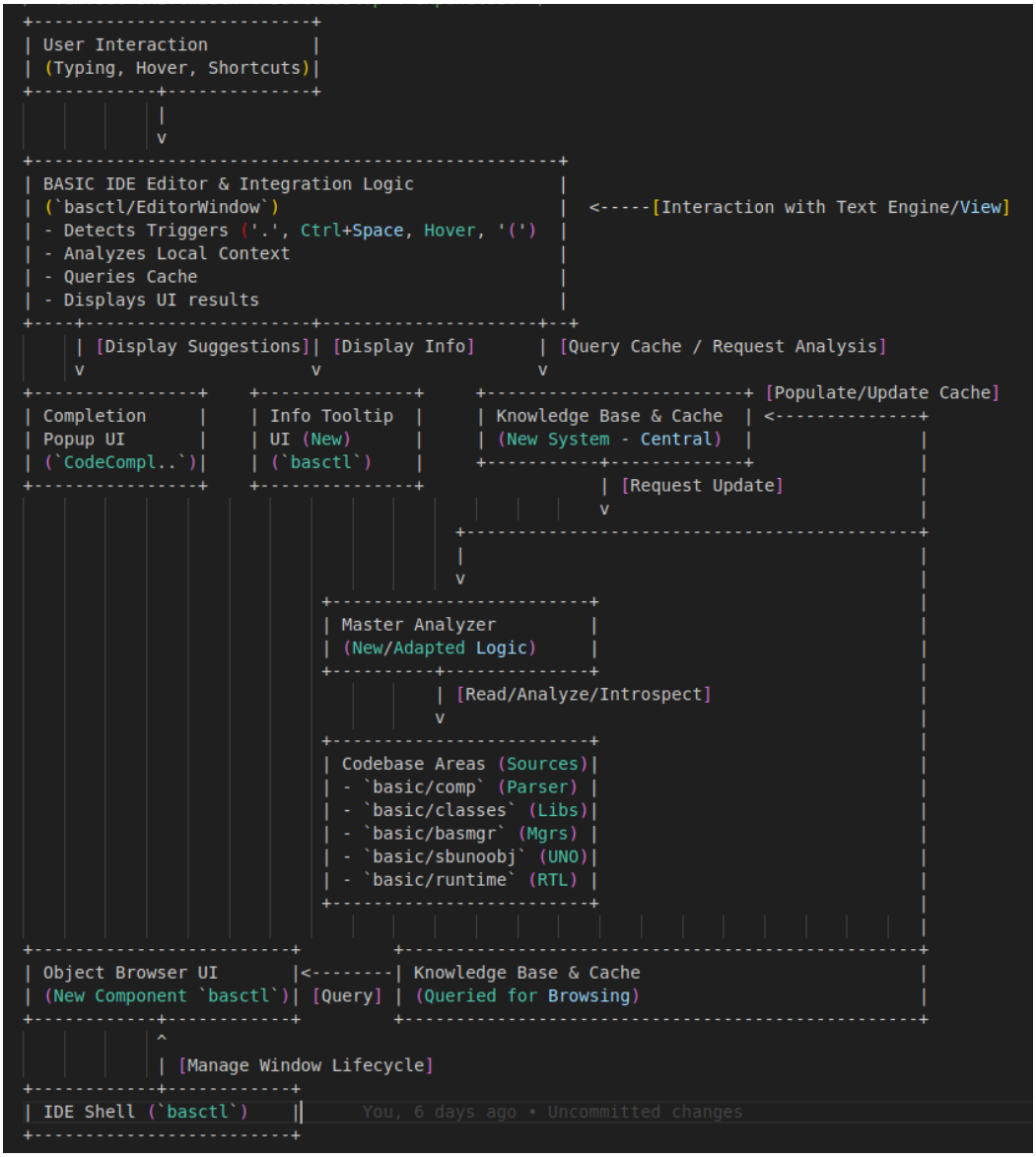
3.6 Component 6: Integration & Configuration Layer

- **Role:** Glues components together, manages UI visibility, handles default settings.
- **Functionality:**
    - Coordinate interactions between Editor, Analyzer, Cache, and UI popups.
    - Manage Object Browser window state via basctl/Shell.
    - Update configuration to enable features by default.
- **Implementation Notes:** Modifications primarily in basctl/ (Shell, EditorWindow) and configuration files/code (potentially related to basic/CodeCompleteOptions).

Part 4: Codebase Mapping & Implementation Details

4.1 Overview

This section connects the architectural components (Master Analyzer, Knowledge Base & Cache, UI Components) to specific areas and key files within the LibreOffice source code (core/). It outlines how existing structures will be leveraged and identifies where new development or significant modification is required.



Technical block diagram visualizing the software architecture for an enhanced code completion system within the LibreOffice BASIC IDE.

4.2 Area 1: BASIC Core Engine (basic/source/) Modifications

- **4.2.1 Library & Object Management (basic/source/basmgr/, basic/source/uno/)**
    - **Existing:** BasicManager (basmgr.cxx) manages libraries; BasicManagerRepository (basicmanagerrepository.cxx) tracks managers; UNO containers (scriptcont.cxx, dlgcont.cxx) provide UNO interfaces and event listening.
    - **Work Needed (Analyzer & Cache Input):** Add mechanisms to BasicManager for the Master Analyzer to query available libraries and access StarBASIC instances efficiently at IDE time. Cache needs to store library metadata. *(Effort: Integration Work)*
- **4.2.2 Runtime Classes & UNO Integration (basic/source/classes/)**
    - **Existing:** StarBASIC (sb.cxx), SbModule (sbxmod.cxx), SbMethod represent runtime structures. SbUnoObject (sbunoobj.cxx) handles UNO interaction/introspection. codecompletecache.cxx is the current limited cache; CodeCompleteOptions reads settings.
    - **Work Needed (Analyzer, Cache, Integration):**
        - Analyzer needs to query StarBASIC/SbModule members.
        - SbUnoObject introspection results must feed the new cache.
        - **CodeCompleteDataCache will be replaced** by the new **Knowledge Base & Cache** system. This new system requires sophisticated data structures for detailed symbol info (signatures, scopes, types) and efficient indexing.

          (Effort: Major Effort / Core Development)
        - CodeCompleteOptions will be modified later to enable features by default.
- **4.2.3 Compiler & Parser (basic/source/comp/)**
    - **Existing:** Full compilation suite: SbiScanner, SbiTokenizer, SbiParser, SbiSymPool, SbiSymDef. Used by SbModule::Compile( ).
    - **Work Needed (Master Analyzer - Major Enhancement):**
        - **Adapt/Extend parsing logic:** Modify SbiParser and symbol table (SbiSymPool/SbiSymDef) or create parallel structures to perform incremental, error-tolerant analysis of editor content.
        - **Extract Detailed Info:** Enhance analysis to capture full signatures, resolve types (including UDTs and UNO types declared with Dim As), determine precise scopes.
        - **Feed Cache:** Continuously update the Knowledge Base & Cache.

          (Effort: Major Effort / Core Development)
- **4.2.4 Runtime Engine & Variables (basic/source/runtime/, basic/source/sbx/)**
    - **Existing:** Execution engine (runtime.cxx), built-in functions (methods*.cxx), variable type representations (sbx*.cxx).
    - **Work Needed (Analyzer Input):** Ensure Analyzer has access to definitions (signatures) of all built-in functions/properties (e.g., via SbRtl symbols) for storage in the cache.

      (Effort: Minor / Data Extraction)

4.3 Area 2: BASIC IDE User Interface (basctl/) Modifications & Additions

- **4.3.1 IDE Shell & Window Management (basidesh.cxx, layout.cxx, bastypes.cxx, scriptdocument.cxx, etc.)**
    - **Existing:** Shell manages IDE state; Layout classes manage docking; ScriptDocument abstracts library access.
    - **Work Needed (Integration Layer):** Integrate the Object Browser window into the Shell and Layout. ScriptDocument will be used by the Analyzer. *(Effort: Integration Work)*

- **4.3.2 Editor Window (baside2b.cxx - EditorWindow)**
    - **Existing:** Hosts text engine, handles input, basic syntax highlighting, contains CodeCompleteWindow, UnoTypeCodeCompletetor, and stubs like HandleCodeCompletion.
    - **Work Needed (Suggestion Assistant, Quick Tip Provider - Significant Modification & UI):**
        - Implement triggering logic (., Ctrl+Space, hover, ().
        - Add context analysis (interacting with text engine/analyzer).
        - Implement queries to the Knowledge Base & Cache.
        - Adapt CodeCompleteWindow or create new UI for suggestions.
        - Create UI for Quick Info / Parameter Info tooltips.
        - Replace/Integrate UnoTypeCodeCompletetor. *(Effort: Significant Modification)*
- **4.3.3 Object Browser Implementation (basctl/ - New Files)**
    - **Existing:** UI patterns from ObjectCatalog.cxx, PropBrw (propbrw.cxx).
    - **Work Needed (Object Browser UI - New Component):** Create new docking window, views, and logic to query cache and display browsable information with details and search. *(Effort: Significant Development)*

- **4.3.4 Dialog Editor (baside3.cxx, dlged/)**
    - **Existing:** Visual designer UI.
    - **Work Needed (Minimal - Analyzer Input):** Ensure Analyzer can list dialogs/controls for the Object Browser. *(Effort: Minor / Data Extraction)*
- **4.3.5 Accessibility (accessibility/)**
    - **Existing:** Framework.
    - **Work Needed:** Implement accessibility for new UI components (Object Browser, tooltips). *(Effort: Integration Work)*

4.4 Effort Focus & Existing Code Leverage

- **Core Development (Major Effort):**
  - Developing the **Master Analyzer** logic (adapting basic/source/comp/).
  - Implementing the **Knowledge Base & Cache** (replacing basic/source/classes/codecompletecache.cxx).
- **Significant Modification/Development:**
  - Implementing the **Object Browser UI** (basctl/ - new files).
  - Integrating **Completion/Info Tip logic and UI** into EditorWindow (basctl/baside2b.cxx).
- **Integration Work:**
  - Connecting backend components to UI triggers and displays.
  - Integrating the Object Browser into the IDE Shell/Layout.
  - Updating configuration (CodeCompleteOptions).
  - Ensuring accessibility.
- **Leveraged Foundation:**
  - Existing IDE structure (basctl/Shell, Layout, EditorWindow base).
  - Core BASIC objects (StarBASIC, SbModule, SbxVariable).
  - Library management (BasicManager, ScriptDocument).
  - UNO integration (SbUnoObject).
  - Existing parser structures (as a base for adaptation).

Part 5: Project Timeline and Milestones

5.1 Overall Approach: Backend First, UI Follows

This timeline outlines a structured approach to completing the project within the standard GSoC coding period (~12-13 weeks). The strategy prioritizes establishing a robust backend (analysis engine and cache) first, as all user-facing features depend on it. This allows for early validation of the core data gathering and provides a solid foundation before implementing the visual components.

Communication Plan: Consistent communication with the mentor is crucial. I plan for daily synchronization via email (summarizing progress and next steps) and aim for brief weekly or bi-weekly check-in meetings (via IRC or other preferred channel) to discuss progress, challenges, and ensure alignment.

5.2 Phase 1: Core Analysis Engine & Cache Implementation (Weeks 1-6: Approx. May 27 - July 7)

- **Goal:** To design and implement the core backend components: the "Master Analyzer" capable of incremental code analysis and the "Knowledge Base & Cache" to store and provide the gathered information efficiently.
- **Key Tasks:**
    - *(Weeks 1-2)*: Finalize the detailed design for incremental analysis (adapting SbiParser/SbiSymPool or parallel structures). Design and implement core data structures and interfaces for the new Knowledge Base & Cache (replacing CodeCompleteDataCache), defining storage needs (signatures, types, scopes, etc.). Confirm query APIs for BasicManager/StarBASIC/SbModule/SbUnoObject.
    - *(Weeks 3-4)*: Implement incremental parsing/analysis logic triggered by editor context. Extract key symbol information (variables, types, basic signatures). Implement querying of libraries/modules. Begin populating the cache.
    - *(Weeks 5-6)*: Integrate UNO introspection results into the cache (using pre-computation/caching for known types). Implement cache update/invalidation mechanisms tied to code changes. Develop backend tests for data extraction accuracy (variables, simple subs, UDTs, basic UNO types).
- **Deliverable/Milestone:** A functional backend system. The cache can be programmatically queried to retrieve information about libraries, modules, basic procedures, globals, and types within the current scope. Demonstrates basic updates based on simulated code changes.

5.3 Phase 2: Object Browser and Core Completion UI (Weeks 7-11: Approx. July 8 - August 11)

- **Goal:** To implement the two major user-facing features: the Object Browser and the primary code completion mechanisms (Ctrl+Space and Dot), building upon the Phase 1 backend.
- **Key Tasks:**
    - *(Weeks 7-8)*: **Object Browser (tdf#165785):** Implement the Object Browser docking window UI within basctl/. Connect UI to query the cache for populating hierarchical views (Libraries -> Modules -> Members). Display basic details for selected items. Integrate into Shell.

    - *(Weeks 9-10)*: **Core Code Completion (tdf#165786, tdf#66185):** Modify EditorWindow (basctl/baside2b.cxx) to handle Ctrl+Space and . triggers. Implement context analysis. Query cache for suggestions. Adapt CodeCompleteWindow to display results (keywords, variables, simple members).
    - *(Week 11)*: **Refine Backend for Features:** Enhance Analyzer and Cache to handle detailed signatures (parameters) and type information necessary for richer completion and upcoming info tips. Test completion with UDTs and more UNO types.
- **Mid-term Milestone (Around Week 7/8 - July 22-28):** Functional backend cache populated. Demonstrable Object Browser showing basic structure. Core Ctrl+Space and . completion functional for basic scenarios.

5.4 Phase 3: Info Tips, Integration, Testing & Polish (Weeks 12-13: Approx. August 12 - August 25)

- **Goal:** Complete all features, ensure robust integration, perform thorough testing, optimize performance, and finalize the codebase for submission.
- **Key Tasks:**
    - *(Week 12)*: **Info Tips (tdf#92253):** Implement Quick Info (hover) and Parameter Info ((trigger) logic in EditorWindow. Create UI elements (tooltips) to display type/signature information queried from the cache.
    - *(Week 13)*: **Integration, Testing, Polish & Default Setting (tdf#165780):**
        - Conduct extensive testing across diverse code examples and scenarios.
        - Fix bugs from integration and testing.
        - Profile and optimize performance of analysis, cache lookups, and UI responsiveness.
        - Refine UI/UX based on usability.
        - Ensure code quality, add comments, update developer documentation/wiki.
        - Implement configuration change (e.g., via CodeCompleteOptions) to enable features by default.
        - Verify accessibility of new UI components.
- **Final Deliverable/Milestone:** All specified features implemented, integrated, tested, reasonably optimized, and enabled by default. Code submitted to Gerrit.

5.5 Phase 4: Final Submission & Wrap-up (Week of August 26)

- **Goal:** Complete all GSoC program requirements.
- **Tasks:**
    - Address final code review feedback on submitted patches.
    - Submit final GSoC project summary and evaluations.
    - Ensure all project documentation is complete and accessible.

Part 6: Conclusion: Value and Impact

6.1 A Foundational Improvement for BASIC Development

Implementing the enhanced code completion, object browser, and inline information features represents a fundamental upgrade to the LibreOffice BASIC IDE. It moves beyond providing basic editing capabilities to offering intelligent assistance that actively supports the developer's workflow. This project addresses long-standing community requests and aims to make BASIC scripting within LibreOffice significantly more productive, accessible, and reliable.

6.2 Key Benefits Summarized

The successful completion of this project will deliver tangible benefits:

- **Increased Productivity:** Reduced time spent on manual lookups, syntax memorization, and correcting typos translates directly to faster development cycles for macros and extensions. Features like context-aware completion (Ctrl+Space) and Parameter Info streamline the coding process.
- **Reduced Errors:** Accurate suggestions minimize typographical errors in keywords, variables, and member names. Parameter Info helps prevent incorrect function/subroutine usage. This leads to more robust and maintainable code.
- **Improved Discoverability:** The Object Browser provides an essential tool for exploring the available BASIC commands and the extensive UNO API, encouraging users to leverage more of LibreOffice's capabilities.
- **Lowered Learning Curve:** Intelligent assistance makes learning LibreOffice BASIC and its associated APIs less daunting for new users, offering an experience closer to modern development environments.
- **Enhanced User Experience:** Replacing the limited, experimental completion feature with a comprehensive, reliable, and default-enabled system significantly improves the overall usability and professional feel of the BASIC IDE.

6.3 Addressing Community Needs

This work directly responds to the needs expressed by the LibreOffice community through various Bugzilla tickets (tdf#165785, tdf#165786, tdf#66185, tdf#92253, tdf#165780), demonstrating a commitment to enhancing the tools used for extending LibreOffice.

6.4 Future Potential

The core analysis engine and knowledge cache developed for this project could serve as a foundation for future IDE enhancements, such as improved refactoring tools, static code analysis capabilities ("linting"), or deeper debugger integration.

By providing a significantly more powerful and helpful development environment, this project empowers users to more effectively automate tasks and extend LibreOffice, ultimately adding substantial value to the entire suite.

About Me

My academic journey began with a fascination for astrophysics, but my path ultimately led me to Computer Science and Engineering(long story), which I completed in 2019. Throughout my undergraduate studies, my innate curiosity drove me to explore diverse fields, including Indian scriptures, human psychology, and the historical influence of various belief systems on societal choices.

This curiosity eventually brought me to the LibreOffice community in Dec, 2023. While initially seeking to gain practical programming experience, I was encouraged by Ilmari Lauhakangas to participate in Google Summer of Code 2024, an opportunity I had considered back in 2017. My GSoC 2024 project involved adding Histogram chart functionality to LibreOffice Calc.

My experience with LibreOffice, particularly during GSoC, has been transformative. Working within such a large and complex codebase taught me invaluable lessons beyond just coding.

I learned the *critical importance of structured thinking*, *clear communication* (especially knowing when to ask for help after dedicated effort, a lesson learned the hard way), and the strategic challenge of identifying the precise location for implementing changes – Kevin Spacey's character in "House of Cards" wasn't wrong about *"location, location, location"* applying broadly, even to software architecture!

My proposal last year was conceptually framed around **entropy**; this year reflects my growth, focusing on **First Principles** thinking.

It mirrors the journey from chaotic initial thoughts to a structured, logical development plan, which I believe is essential for tackling a project of this scope and difficulty successfully.

https://github.com/varshneydevansh - *My GitHub profile*

https://gerrit.libreoffice.org/q/devansh+varshney - My pending and accepted PRs for the LibreOffice

- My Socials

https://x.com/varshneydevansh
https://www.youtube.com/@kyuantym
https://www.youtube.com/@varshneydevansh
https://devanshvarshney.blogspot.com/

Additional resources used to build this report

https://help.libreoffice.org/latest/en-US/text/shared/optionen/BasicIDE.html?DbPAR=BASE&System=WIN

https://documentation.libreoffice.org/en/english-documentation/macro/

https://books.libreoffice.org/en/GS248/GS24801-LOBasics.html

----------END of Report----------