STUNIR Phase 2 API Reference

Build System & Configuration

Epoch_Types

Types

```
type Epoch_Value is range 0 .. 2**63 - 1;

type Epoch_Source is (
   Source_Unknown,
   Source_Env_Build_Epoch,      --  STUNIR_BUILD_EPOCH
   Source_Env_Source_Date,      --  SOURCE_DATE_EPOCH
   Source_Derived_Spec_Digest,  --  Derived from spec/ tree
   Source_Git_Commit,           --  From git log
   Source_Zero,                 --  Fallback to 0
   Source_Current_Time          --  Non-deterministic
);

type Epoch_Selection is record
   Value           : Epoch_Value := 0;
   Source          : Epoch_Source := Source_Unknown;
   Is_Deterministic : Boolean := True;
   Spec_Digest     : Hash_Hex := Zero_Hash;
end record;
```

Functions

```
function Source_To_String (S : Epoch_Source) return String;
function String_To_Source (S : String) return Epoch_Source;
function Is_Deterministic_Source (S : Epoch_Source) return Boolean;
```

Epoch_Selector

Procedures

```
procedure Select_Epoch (
   Spec_Root     : Path_String;
   Allow_Current : Boolean := False;
   Selection     : out Epoch_Selection);
```

Selects epoch using priority:

1. `STUNIR_BUILD_EPOCH` environment variable
2. `SOURCE_DATE_EPOCH` environment variable

3. Derived from spec directory digest
4. Zero (fallback)

Functions

```
function Parse_Epoch_Value (S : String) return Epoch_Value;
function Compute_Spec_Digest (Spec_Root : Path_String) return Hash_Hex;
function Derive_Epoch_From_Digest (Digest : Hash_Hex) return Epoch_Value;
function To_JSON (Selection : Epoch_Selection) return Epoch_JSON;
```

Toolchain_Types

Types

```
type Tool_Requirement is (Required, Optional);

type Tool_Status is (
   Status_Unknown,
   Status_Resolved,
   Status_Not_Found,
   Status_Hash_Mismatch,
   Status_Version_Mismatch
);

type Tool_Entry is record
   Logical_Name   : Short_String;
   Binary_Name    : Short_String;
   Resolved_Path  : Path_String;
   SHA256_Hash    : Hash_Hex;
   Version_String : Medium_String;
   Requirement    : Tool_Requirement;
   Status         : Tool_Status;
end record;

type Tool_Registry is record
   Entries        : Tool_Vector;
   Count          : Natural;
   Required_Count : Natural;
   Resolved_Count : Natural;
end record;
```

Functions

```
function Is_Resolved (E : Tool_Entry) return Boolean;
function All_Required_Resolved (Reg : Tool_Registry) return Boolean;
function Find_Tool (Reg : Tool_Registry; Name : Short_String) return Natural;
```

Toolchain_Scanner

Types

```
type Builtin_Tool is (
   Tool_Python,
   Tool_Bash,
   Tool_Git,
   Tool_CC,
   Tool_Rustc,
   Tool_Cargo,
   Tool_Stunir_Native
);
```

Procedures

```
procedure Initialize_Registry (Reg : out Tool_Registry);
procedure Add_Tool (
   Reg : in out Tool_Registry;
   Logical_Name, Binary_Name : Short_String;
   Requirement : Tool_Requirement;
   Success : out Boolean);
procedure Resolve_Tool (Tool_Ent : in out Tool_Entry; Success : out Boolean);
procedure Resolve_All (Reg : in out Tool_Registry; Success : out Boolean);
procedure Scan_Toolchain (Lockfile : out Toolchain_Lockfile; Success : out Boolean);
```

Build_Config

Types

```
type Build_Profile is (
   Profile_Auto,
   Profile_Native,
   Profile_Python,
   Profile_Shell
);

type Build_Phase is (
   Phase_Discovery,
   Phase_Epoch,
   Phase_Spec_Parse,
   Phase_IR_Emit,
   Phase_Code_Gen,
   Phase_Compile,
   Phase_Receipt,
   Phase_Verify
);

type Configuration is record
   Profile        : Build_Profile;
   Spec_Root      : Path_String;
   Output_IR      : Path_String;
   Output_Code    : Path_String;
   Lock_File      : Path_String;
   Native_Binary  : Path_String;
   Strict_Mode    : Boolean;
   Verbose        : Boolean;
   Is_Valid       : Boolean;
end record;
```

Procedures

```
procedure Initialize_Config (Config : out Configuration);
procedure Set_Default_Paths (Config : in out Configuration);
```

Functions

```
function Validate_Config (Config : Configuration) return Boolean;
function Profile_To_String (P : Build_Profile) return String;
function String_To_Profile (S : String) return Build_Profile;
function Phase_To_String (P : Build_Phase) return String;
```

Dependency_Types

Types

```
type Dependency_Status is (
   Dep_Unknown,
   Dep_Accepted,
   Dep_Rejected,
   Dep_Not_Found,
   Dep_Version_Mismatch,
   Dep_Hash_Mismatch
);

type Dependency_Kind is (
   Dep_Tool,
   Dep_Library,
   Dep_Module,
   Dep_File
);

type Dependency_Entry is record
   Name            : Short_String;
   Kind            : Dependency_Kind;
   Status          : Dependency_Status;
   Resolved_Path   : Path_String;
   Expected_Hash   : Hash_Hex;
   Actual_Hash     : Hash_Hex;
   Is_Optional     : Boolean;
   Is_Accepted     : Boolean;
end record;
```

Receipt_Types

Types

```
type Receipt_Status is (
   Receipt_Created,
   Receipt_Skipped_No_Compiler,
   Receipt_Skipped_No_Source,
   Receipt_Binary_Emitted,
   Receipt_Compilation_Failed,
   Receipt_Verification_Passed,
   Receipt_Verification_Failed
);

type Build_Receipt is record
   Schema       : Short_String;
   Kind         : Receipt_Kind;
   Target       : Path_String;
   Target_Hash  : Hash_Hex;
   Status       : Receipt_Status;
   Epoch        : Epoch_Value;
   Tool         : Tool_Info;
   Inputs       : Input_File_Vector;
   Input_Count  : Natural;
   Is_Valid     : Boolean;
end record;
```

Build_Orchestrator

Types

```
type Build_Result is record
   Status        : Build_Status;
   Final_Profile : Build_Profile;
   Error_Message : Medium_String;
   Phase_Results : Phase_Status_Array;
   Epoch         : Epoch_Selection;
   Receipt_Count : Natural;
end record;
```

Procedures

```
procedure Run_Build (Config : Configuration; Result : out Build_Result);
procedure Execute_Phase (Config : Configuration; Phase : Build_Phase;
                         Result : in out Build_Result);
```

Functions

```
function Detect_Runtime (Config : Configuration;
                         Lockfile : Toolchain_Lockfile) return Build_Profile;
function All_Phases_Succeeded (Result : Build_Result) return Boolean;
```

Usage Example

```
with Build_Config;       use Build_Config;
with Build_Orchestrator; use Build_Orchestrator;

procedure Main is
   Config : Configuration;
   Result : Build_Result;
begin
   Initialize_Config (Config);
   Set_Default_Paths (Config);
   Config.Is_Valid := Validate_Config (Config);

   if Config.Is_Valid then
      Run_Build (Config, Result);

      if Result.Status = Status_Success then
         --  Build succeeded
         null;
      end if;
   end if;
end Main;
```