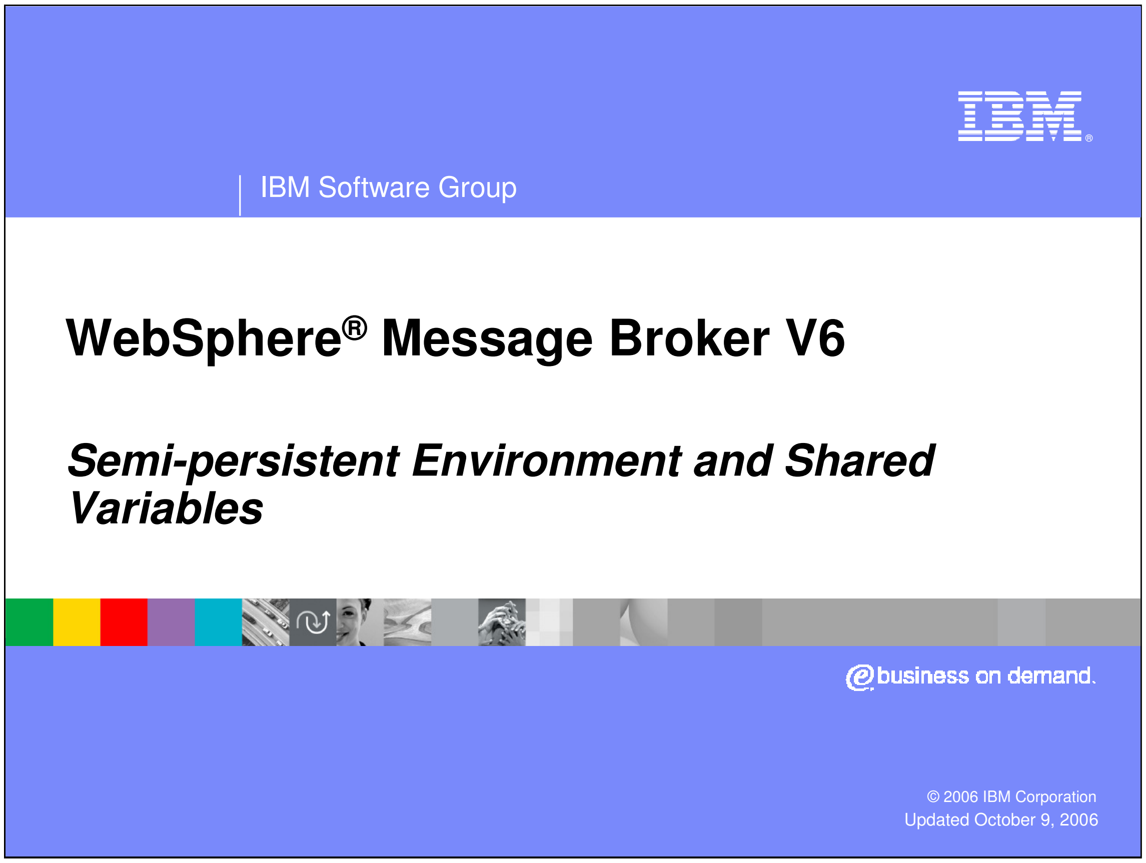


IBM Software Group

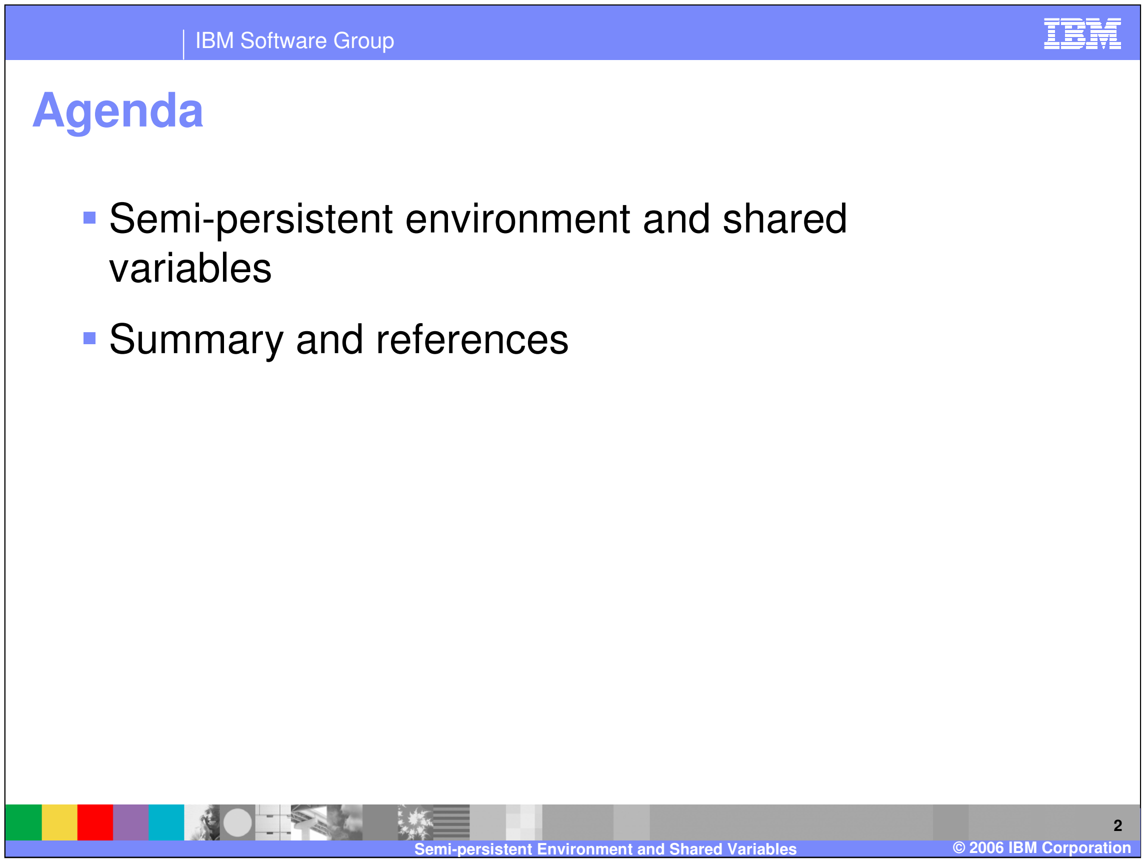
WebSphere® Message Broker V6

Semi-persistent Environment and Shared Variables

business on demand.

Updated October 9, 2006

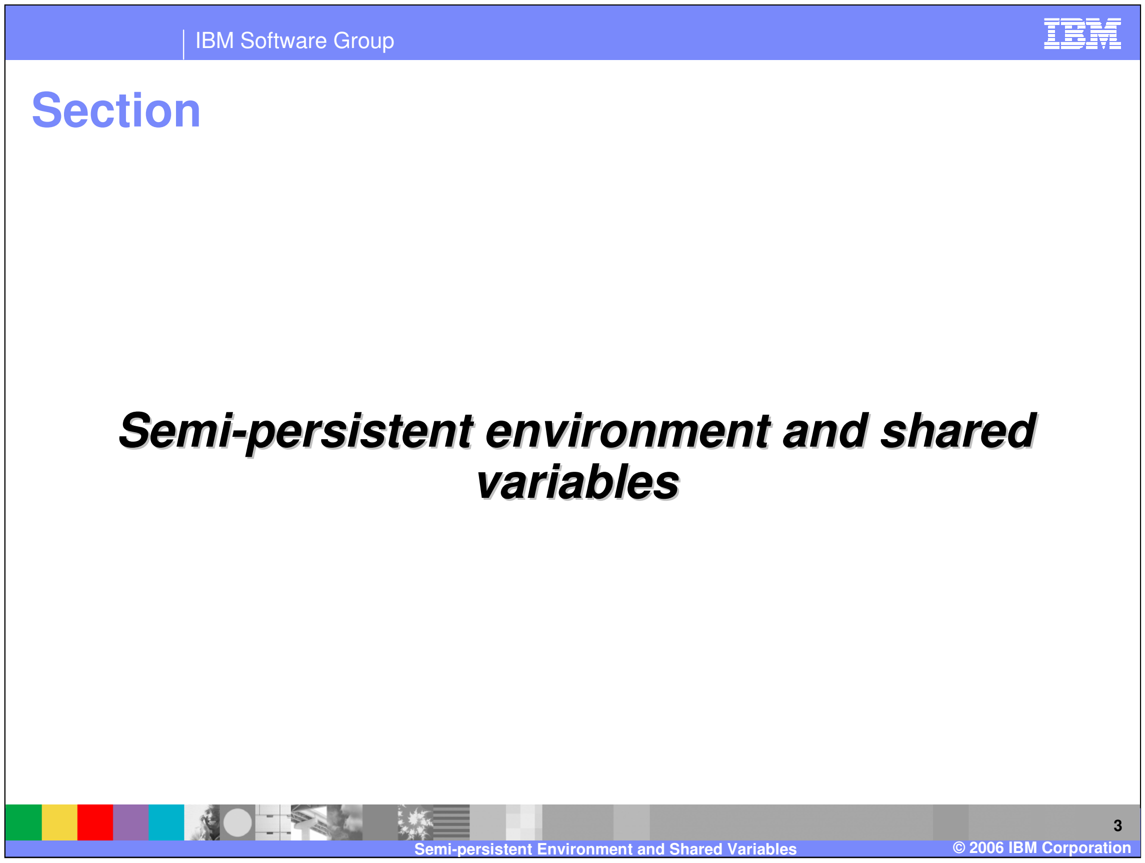
This presentation discusses Semi-persistent Environment and Shared Variables.



Agenda

- Semi-persistent environment and shared variables
- Summary and references

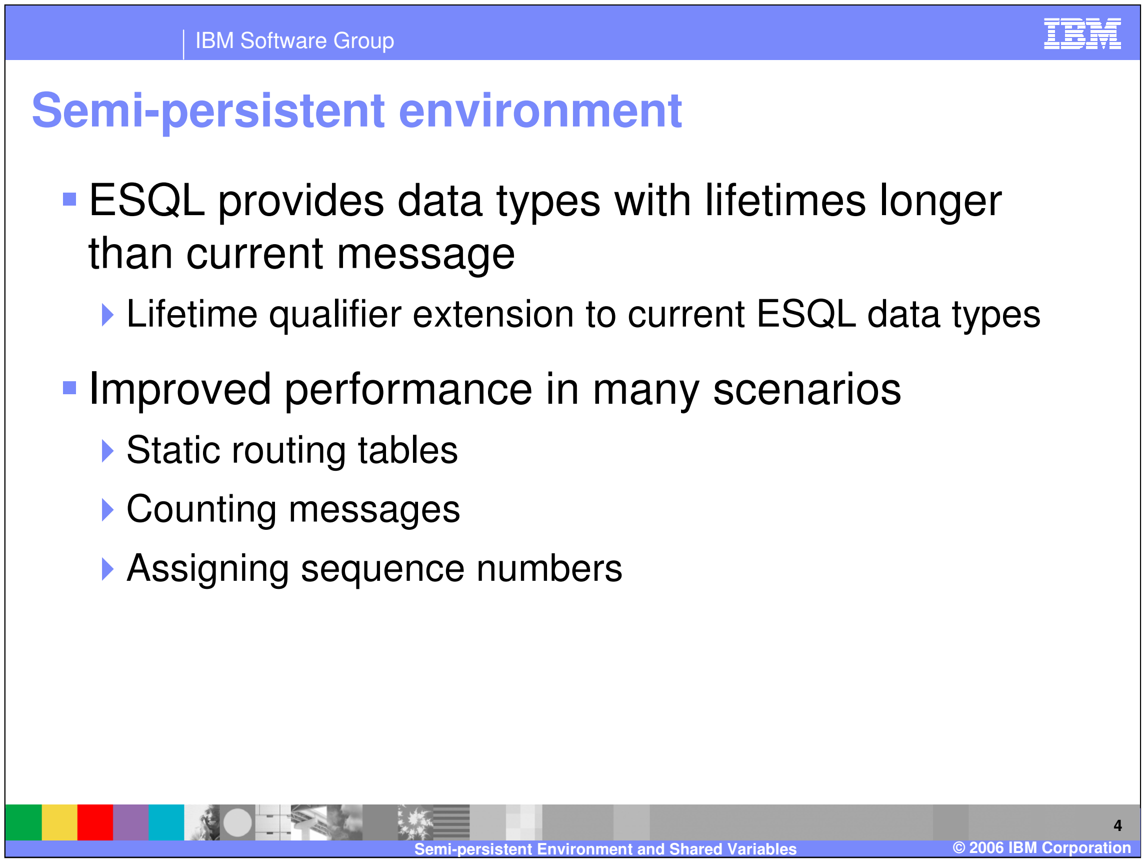
This topic will discuss the new Version 6 Semi-persistent Environment and Shared Variables enhancements.



Section

Semi-persistent environment and shared variables

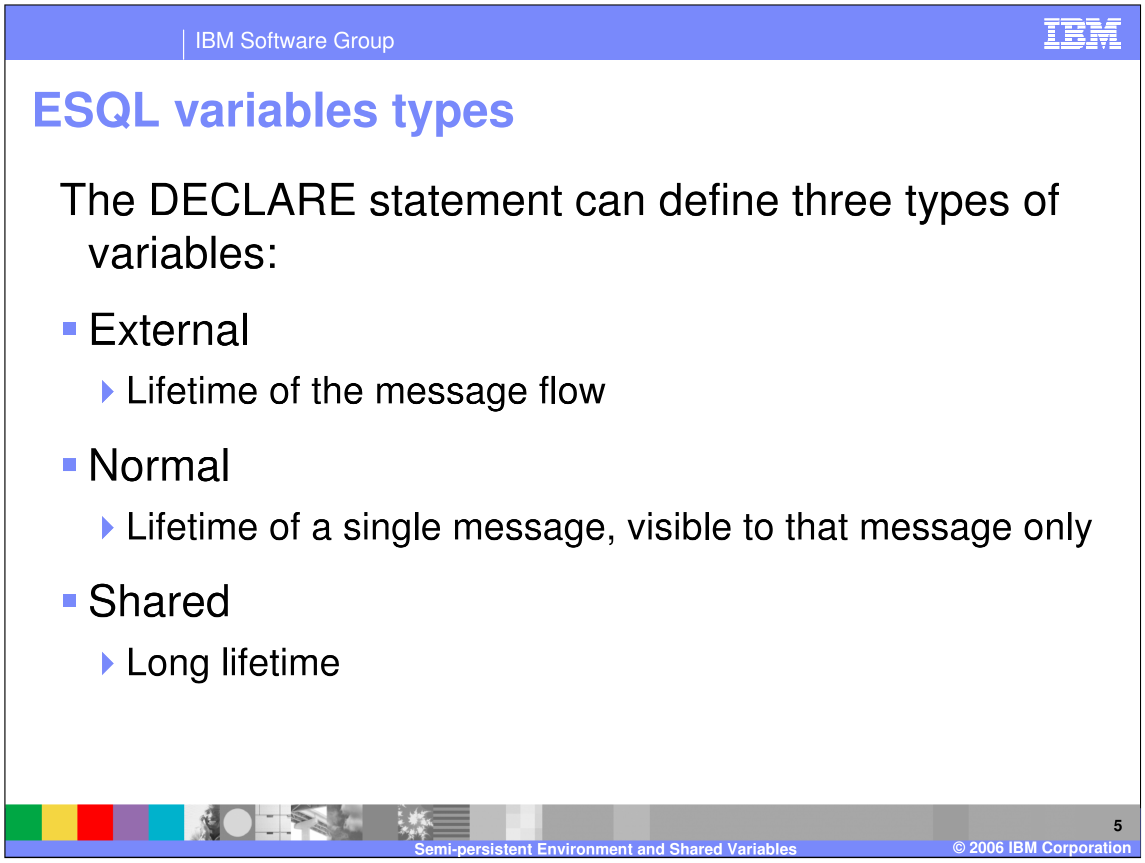
This section provides technical detail about the Version 6 Semi-persistent Environment and Shared Variables.



Semi-persistent environment

- ESQL provides data types with lifetimes longer than current message
  - Lifetime qualifier extension to current ESQL data types
- Improved performance in many scenarios
  - Static routing tables
  - Counting messages
  - Assigning sequence numbers

In previous releases, data types had a lifetime that lasted generally for the period of time that the current message was being processed in one particular node. If you wanted data types to persist longer, you were required to create your own process to do that, which might involve writing the information to a database, reducing efficiency because writing and reading a database is generally a slow process. In Version 6, a semi-persistent environment or “cache” is provided so that data types can now persist across more than one node and longer than the lifetime of the current message. This results in improved performance in many scenarios, especially since database access is not required to retrieve the values for the data types.



ESQL variables types

The DECLARE statement can define three types of variables:

- External
  - Lifetime of the message flow
- Normal
  - Lifetime of a single message, visible to that message only
- Shared
  - Long lifetime

You can use the DECLARE statement to define three types of variables:

External

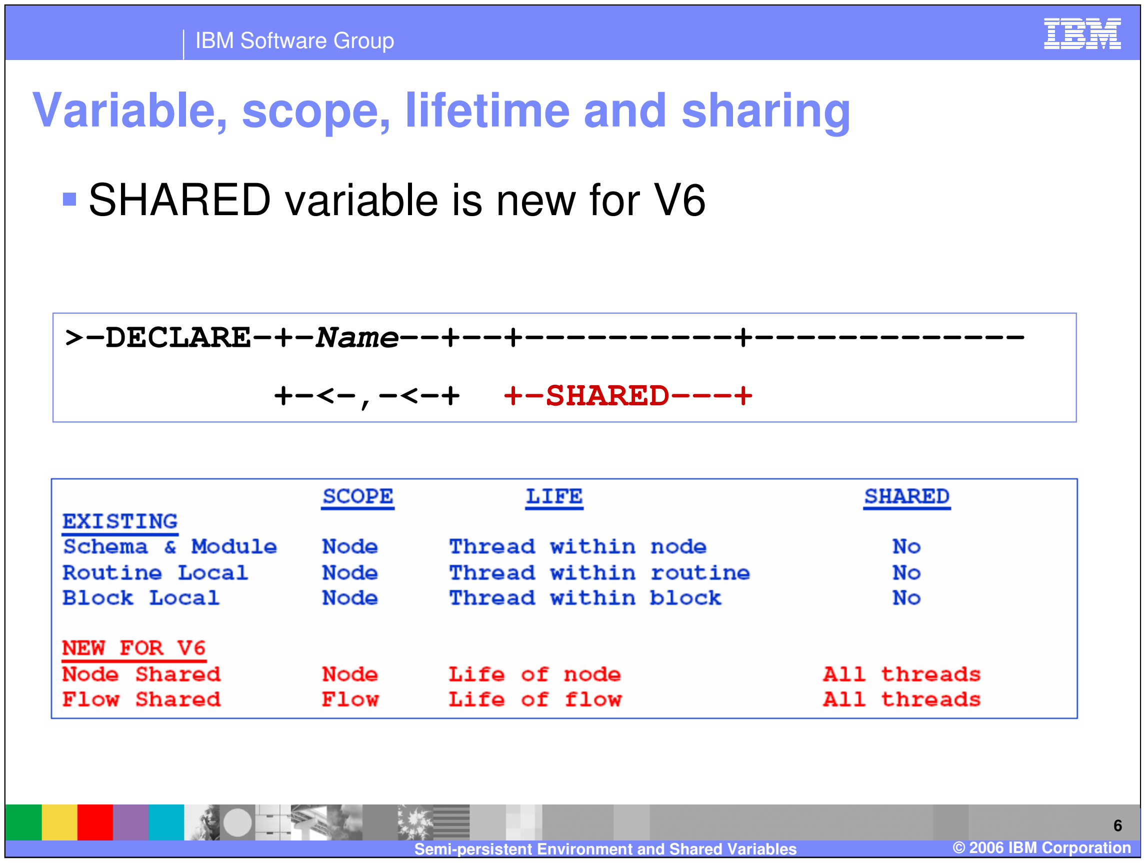
External variables, defined with the EXTERNAL keyword, are also known as *user-defined properties,* or UDPs. They exist for the entire lifetime of a message flow and are visible to all messages passing through the flow. Their initial values, optionally set by the DECLARE statement, can be modified at design time by the Message Flow editor, or at deployment time by the BAR editor. Their values cannot be modified by ESQL.

Normal

"Normal" variables have a lifetime of just one message passing through a node. They are visible to that message only. To define a "normal" variable, omit both the EXTERNAL and SHARED keywords.

Shared

Shared variables can be used to implement an in-memory cache in the message flow. Shared variables have a long lifetime and are visible to multiple messages passing through a flow. They exist for the lifetime of the execution group process, the lifetime of the flow or node, or the lifetime of the node’s ESQL that declares the variable, whichever is shortest. They are initialized when the first message passes through the flow or node after each broker starts up.



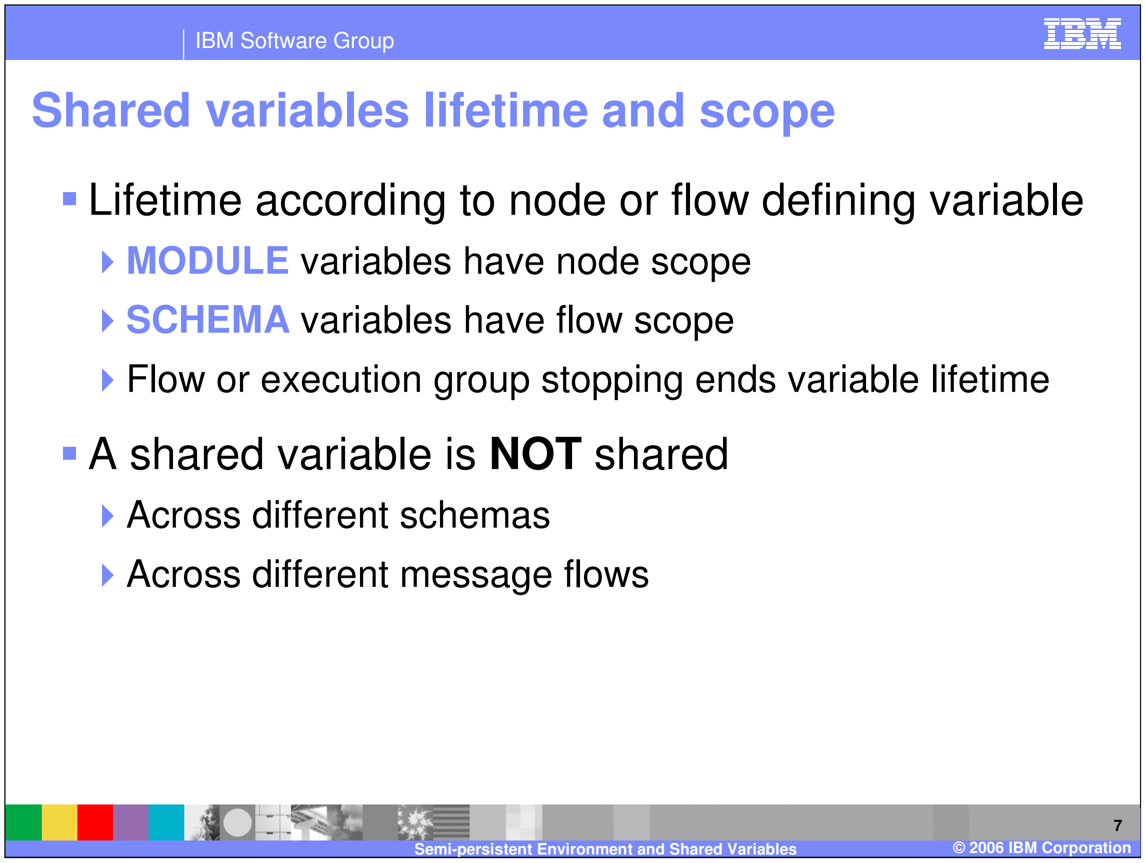
Variable, scope, lifetime and sharing

- SHARED variable is new for V6

```
>-DECLARE-+-Name--+--+----------+------------
           +-<-,-<-+  +-SHARED---+
```

	SCOPE	LIFE	SHARED
EXISTING			
Schema & Module	Node	Thread within node	No
Routine Local	Node	Thread within routine	No
Block Local	Node	Thread within block	No
NEW FOR V6			
Node Shared	Node	Life of node	All threads
Flow Shared	Flow	Life of flow	All threads

The Version 6 SHARED variable uses the semi-persistent environment for storage and access of shared variable data. In the bottom portion of the diagram, you see a summary of the scope, lifetime, and thread visibility of the new SHARED variable. Where you define the variable determines whether the variable is "Node Shared" or "Flow Shared".

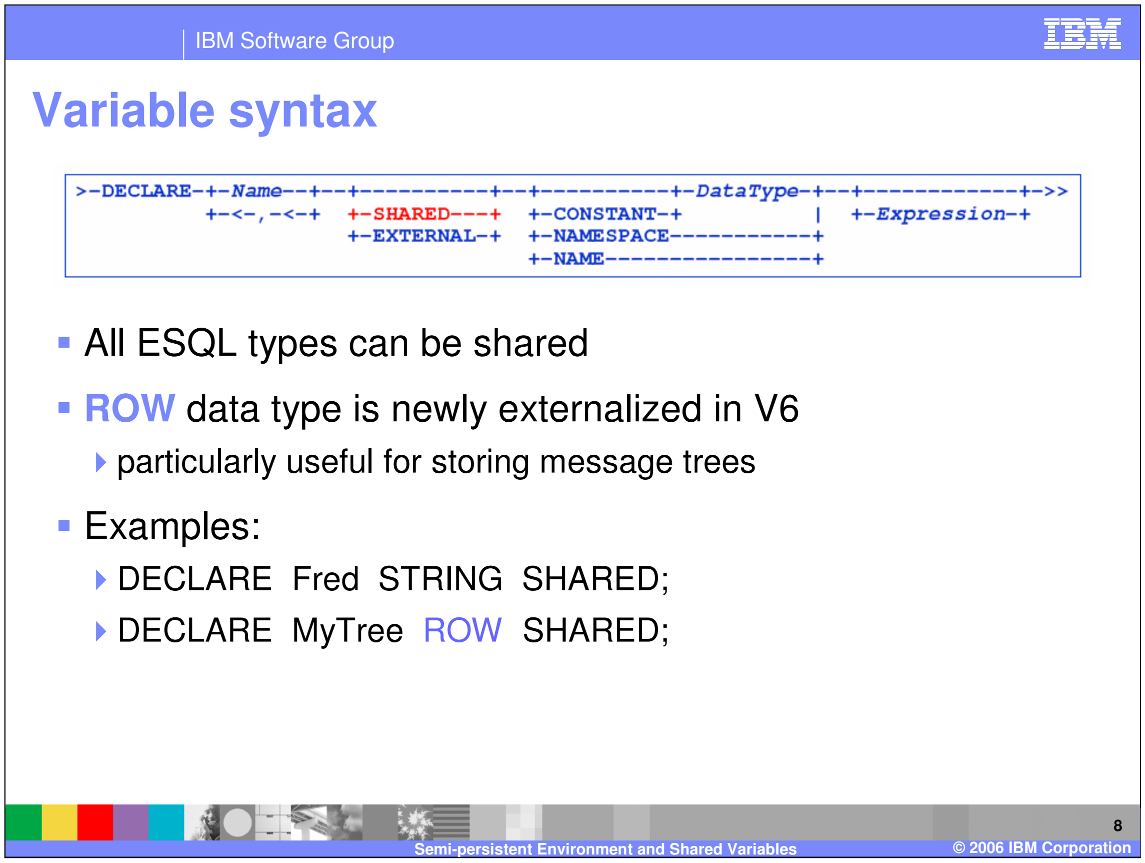


Shared variables lifetime and scope

- Lifetime according to node or flow defining variable
  - **MODULE** variables have node scope
  - **SCHEMA** variables have flow scope
  - Flow or execution group stopping ends variable lifetime
- A shared variable is **NOT** shared
  - Across different schemas
  - Across different message flows

The lifetime and scope of sharing is determined by where the variable is defined. Variables defined as SHARED within modules are “Node Shared” and have node scope for all threads in that node. Variables defined as SHARED within schemas are “Flow Shared” and have flow scope for all threads and all nodes in the message flow. If the flow or execution group is stopped, the variable lifetime ends.

Shared variables are NOT shared if they are defined within different schemas, or if they are defined within different message flows, even if the different message flows are using the same schema.



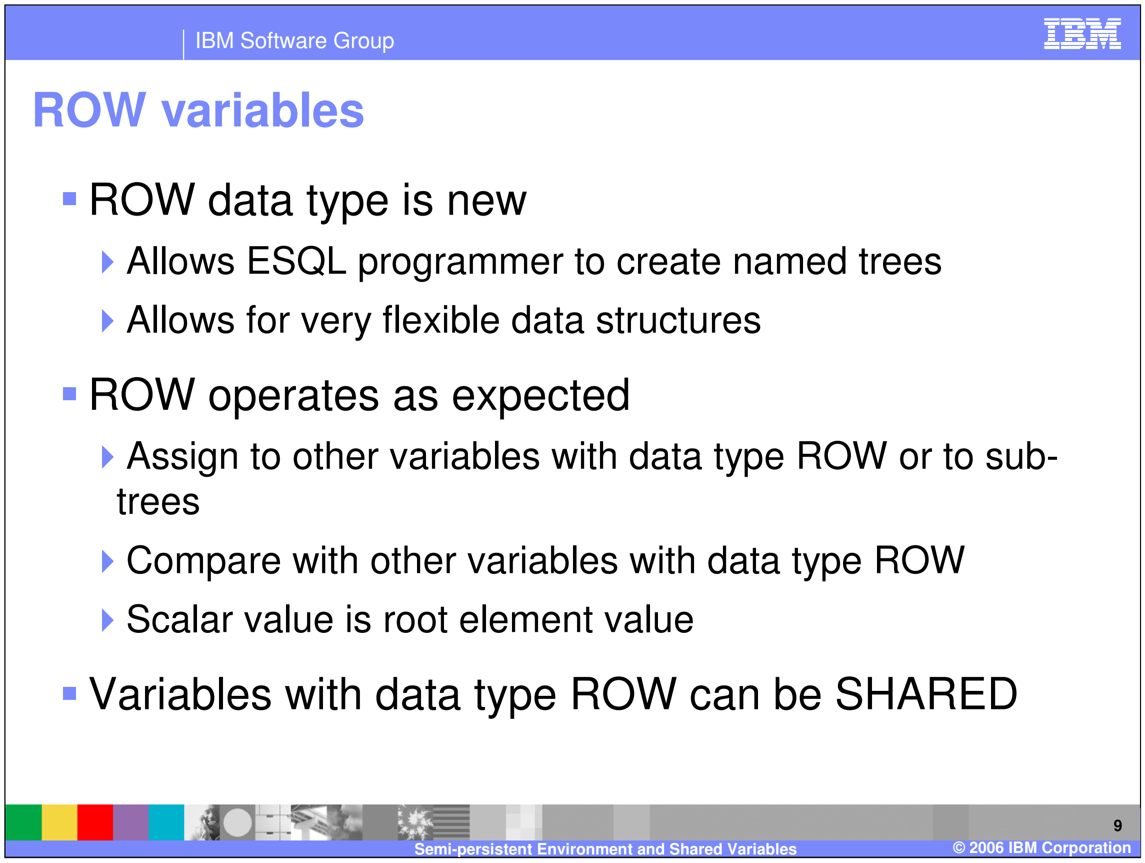
Variable syntax

```
>-DECLARE-+-Name--+--+----------+--+----------+-DataType-+--+------------+->>
          +-<-,-<-+  +-SHARED---+  +-CONSTANT-+          |  +-Expression-+
                     +-EXTERNAL-+  +-NAMESPACE-----------+
                                   +-NAME----------------+
```

- All ESQL types can be shared
- ROW data type is newly externalized in V6
  - particularly useful for storing message trees
- Examples:
  - DECLARE Fred STRING SHARED;
  - DECLARE MyTree ROW SHARED;

This slide summarizes the “DECLARE variable” syntax with emphasis on the new functionality. If neither SHARED nor EXTERNAL is specified, the variable is a normal unshared variable.   A new parameter “SHARED” is provided in V6 to define a shared variable, which is stored in the semi-persistent environment. All ESQL types can be shared.

A newly externalized ROW *DataType* is particularly useful for storing message trees, either as a normal variable or a shared variable.

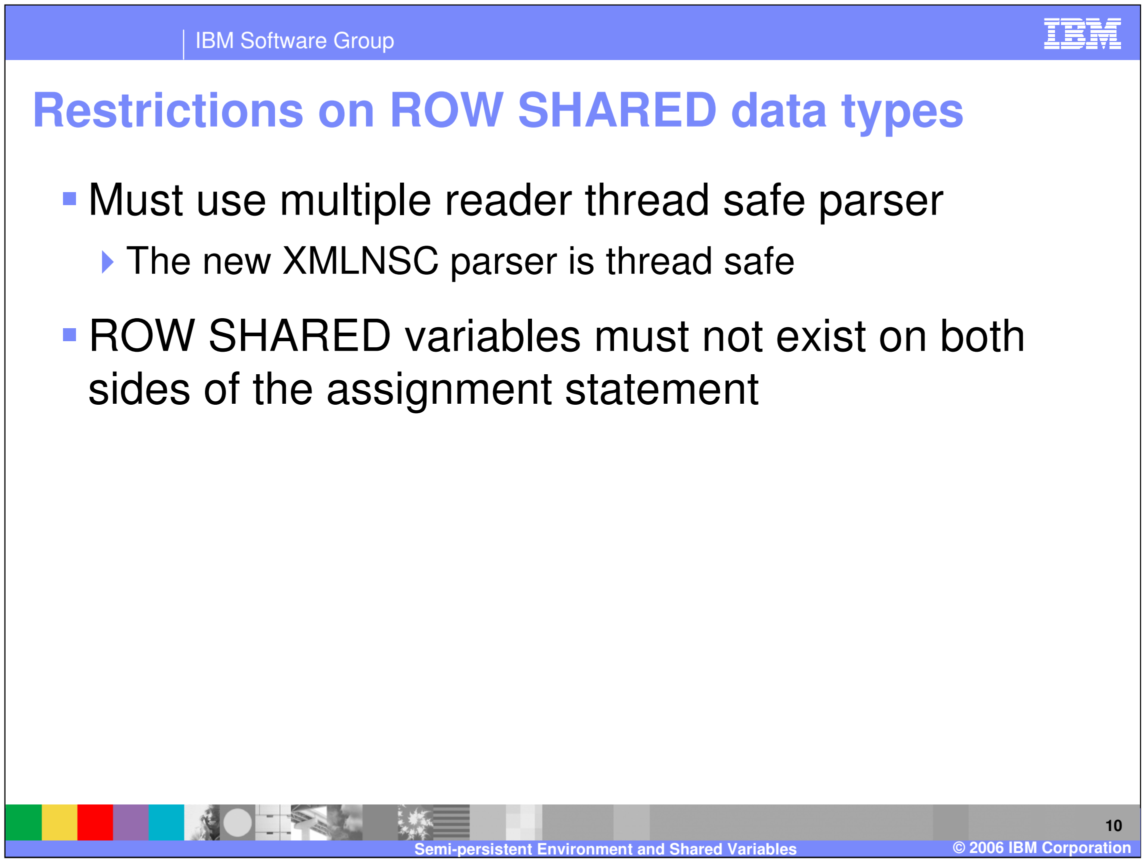


ROW variables

- ROW data type is new
  - Allows ESQL programmer to create named trees
  - Allows for very flexible data structures
- ROW operates as expected
  - Assign to other variables with data type ROW or to sub-trees
  - Compare with other variables with data type ROW
  - Scalar value is root element value
- Variables with data type ROW can be SHARED

The ROW data type is new to V6. It allows you to create an entire named tree with very flexible data structures. Variables defined with the ROW data type operate as you would expect. You can assign the variable to other variables with data type ROW or to sub-trees. You can compare the variable to other variables with data type ROW. The scalar value of a variable defined with data type ROW is its root element value.

If a variable with data type ROW is defined as SHARED, its lifetime, scope and visibility is handled in the same way by the semi-persistent environment as other SHARED variables.

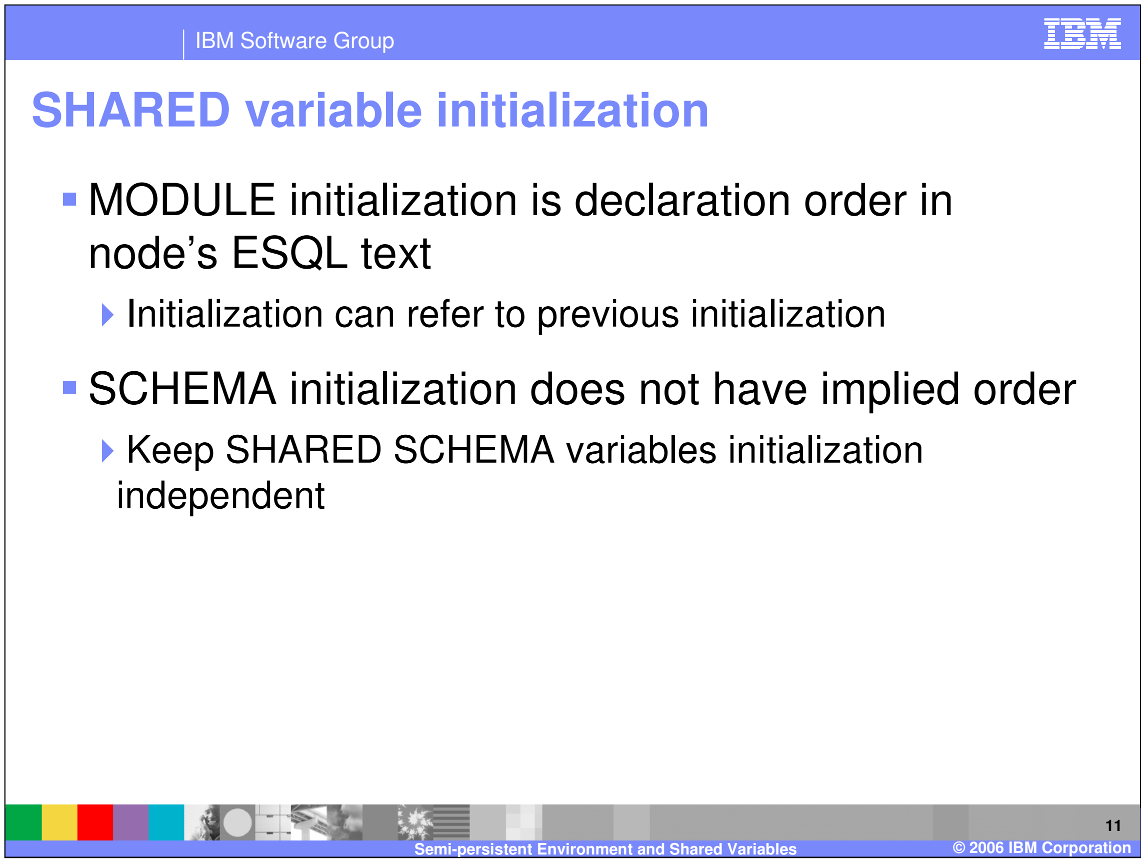


Restrictions on ROW SHARED data types

- Must use multiple reader thread safe parser
  - The new XMLNSC parser is thread safe
- ROW SHARED variables must not exist on both sides of the assignment statement

Shared variables with ROW data types must be parsed with a thread safe parser. The new XMLNSC parser is thread safe and can be used.

ROW SHARED variables must not exist on both sides of an assignment statement. For example, a ROW SHARED variable cannot be assigned to a ROW SHARED variable.

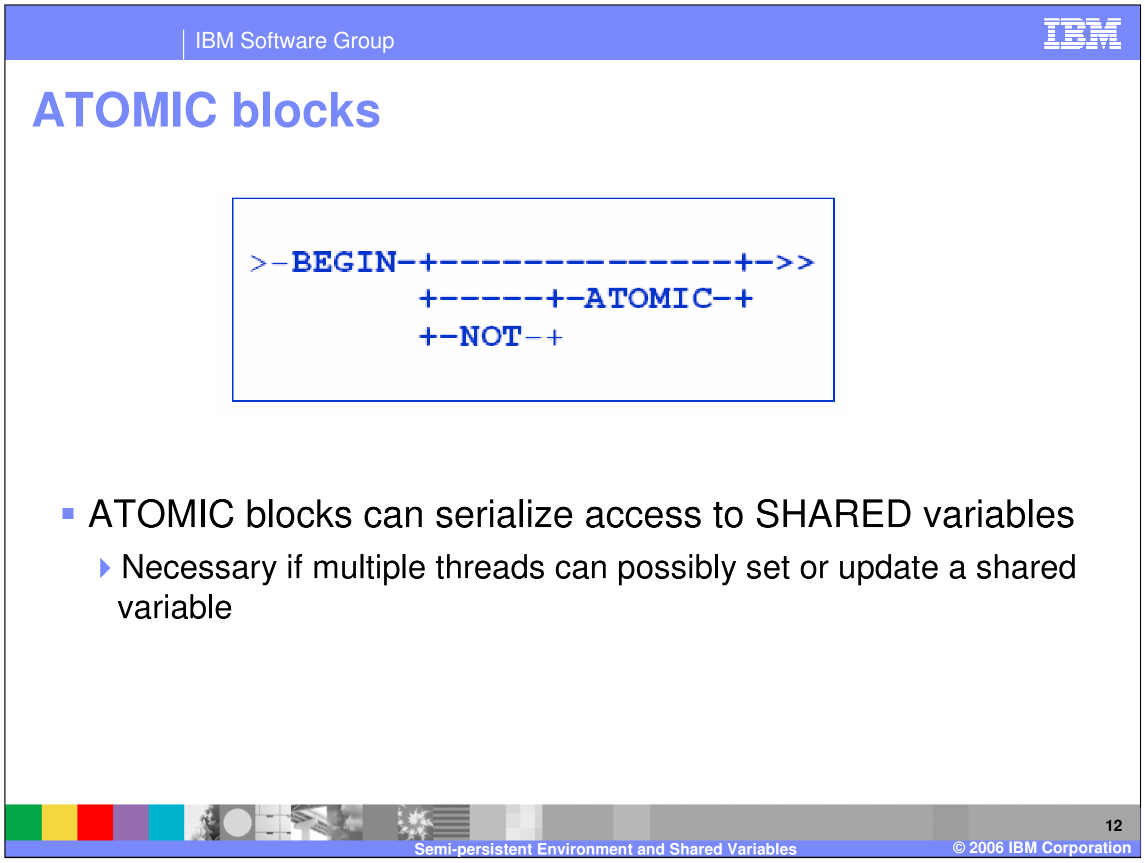


SHARED variable initialization

- MODULE initialization is declaration order in node’s ESQL text
  - Initialization can refer to previous initialization
- SCHEMA initialization does not have implied order
  - Keep SHARED SCHEMA variables initialization independent

The SHARED variables defined within modules are initialized in the order in which they are declared within the node’s ESQL text.

Since schema initialization has no implied order, your application design should allow for this when working with SHARED SCHEMA variables.

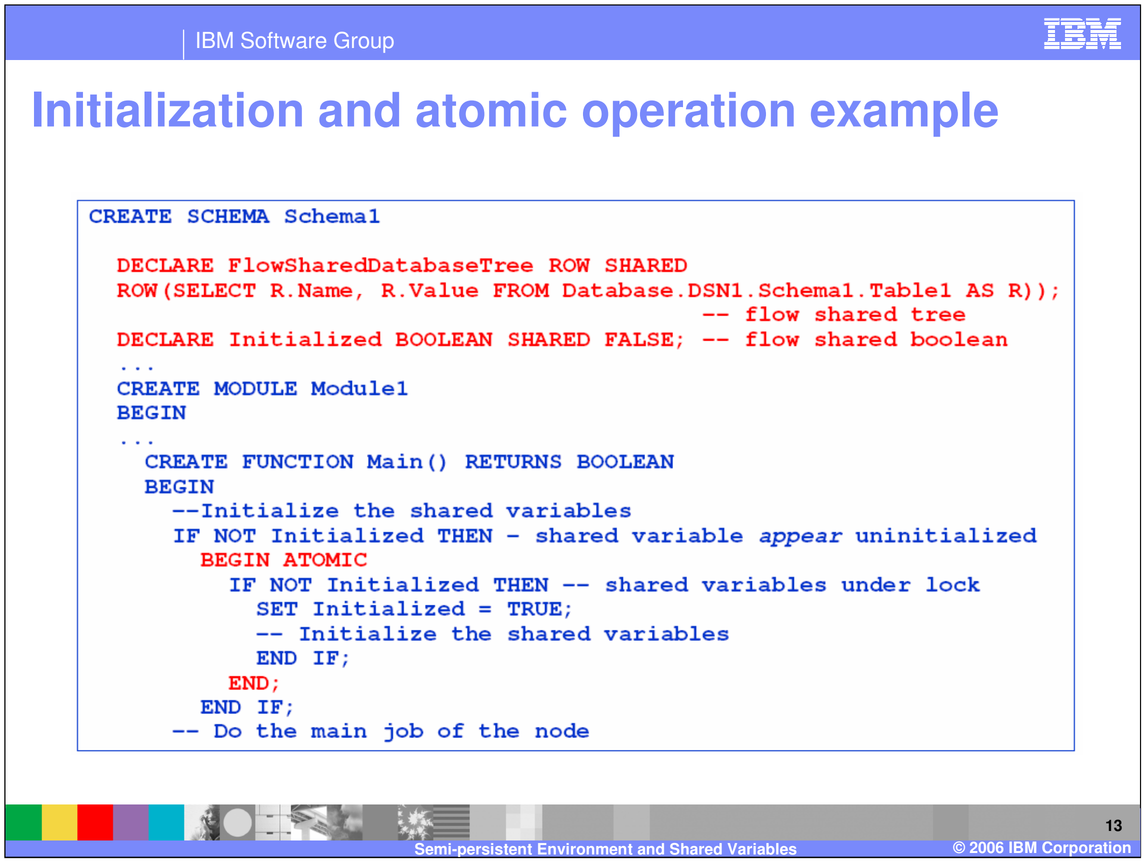


ATOMIC blocks

```
>-BEGIN-+---------------+->>
        +------+-ATOMIC-+
        +-NOT--+
```

- ATOMIC blocks can serialize access to SHARED variables
  - Necessary if multiple threads can possibly set or update a shared variable

Because it is possible that more than one thread can be setting or updating a shared variable, you can ensure the integrity of that shared variable by using an Atomic block to enclose the code that does the set or update of the variable. An Atomic block uses a lock to ensure that only one thread runs within the block at the same time; all other threads are suspended on the lock until the lock owner exits the block. In that way, setting or updating the variable is serialized.

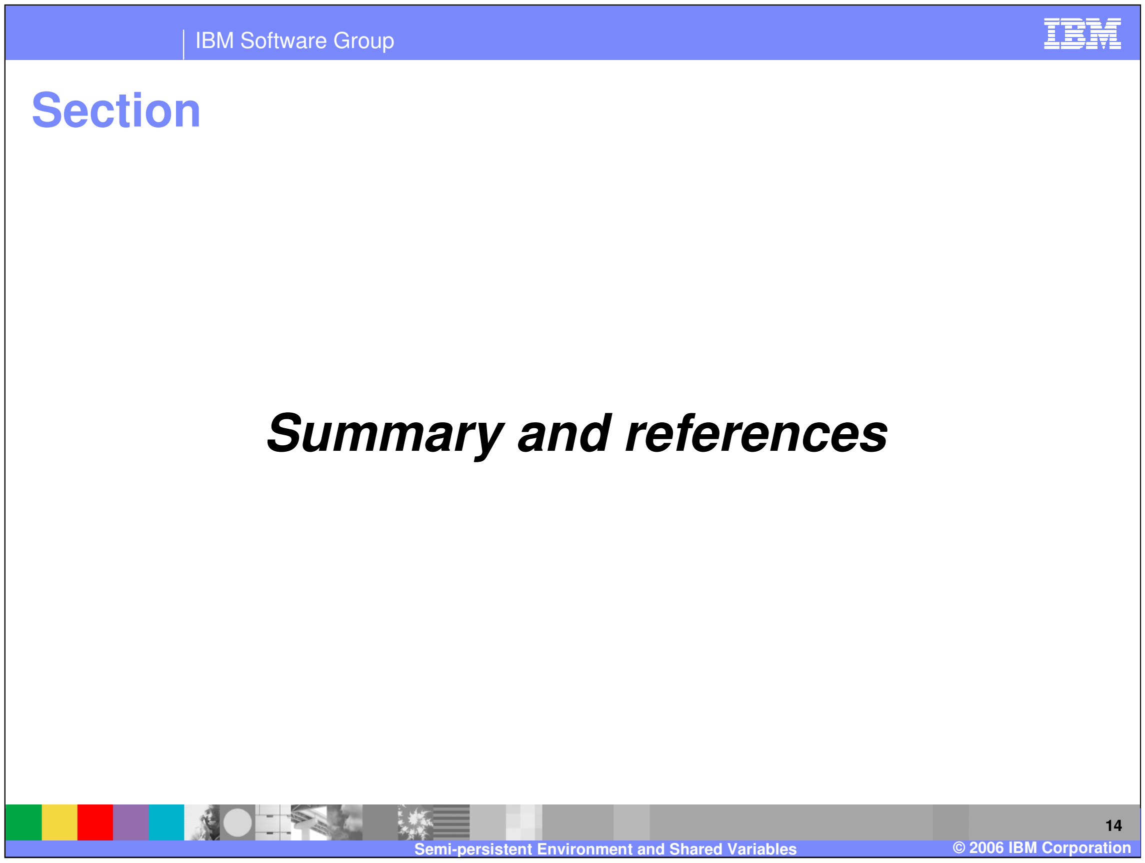


Initialization and atomic operation example

```
CREATE SCHEMA Schema1

  DECLARE FlowSharedDatabaseTree ROW SHARED
  ROW(SELECT R.Name, R.Value FROM Database.DSN1.Schema1.Table1 AS R));
                                             -- flow shared tree
  DECLARE Initialized BOOLEAN SHARED FALSE; -- flow shared boolean
  ...
  CREATE MODULE Module1
  BEGIN
  ...
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
      --Initialize the shared variables
      IF NOT Initialized THEN – shared variable appear uninitialized
        BEGIN ATOMIC
          IF NOT Initialized THEN -- shared variables under lock
            SET Initialized = TRUE;
            -- Initialize the shared variables
            END IF;
          END;
        END IF;
      -- Do the main job of the node
```

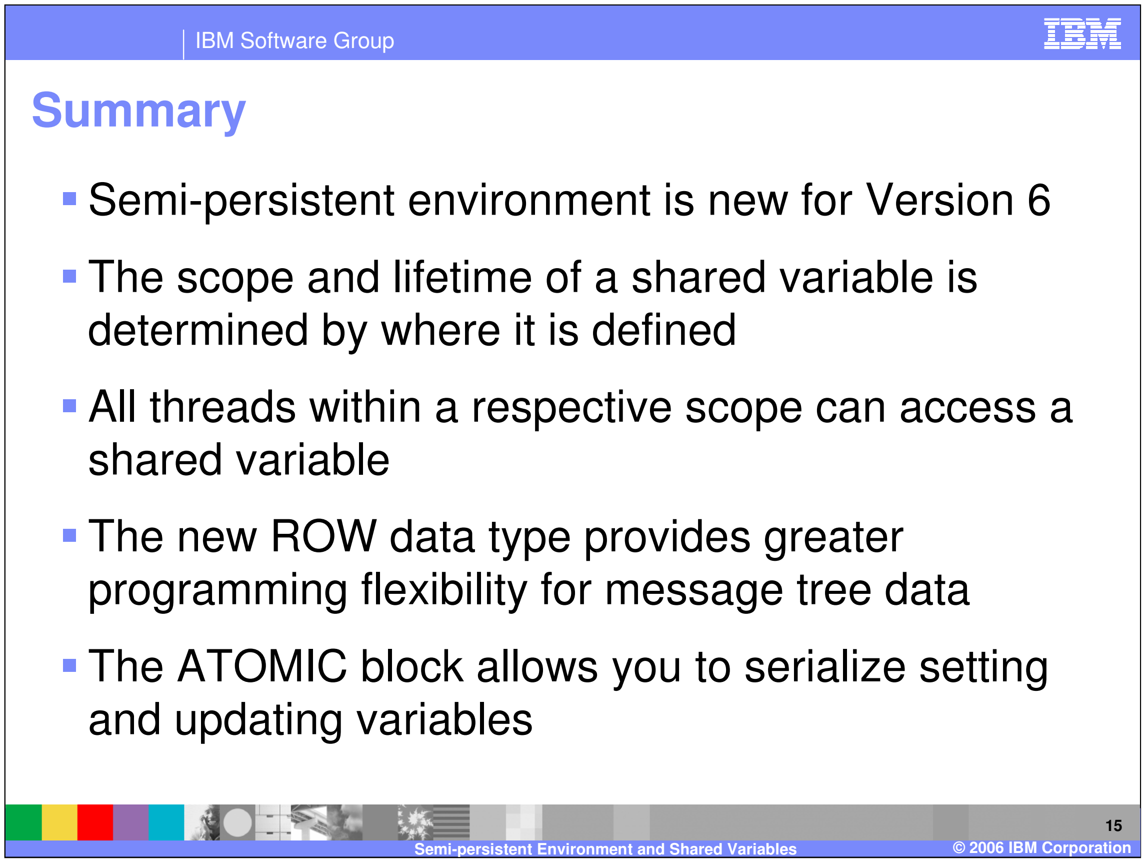
Here is an example of using an ATOMIC block to control the initialization of SCHEMA SHARED variables.



Section

Summary and references

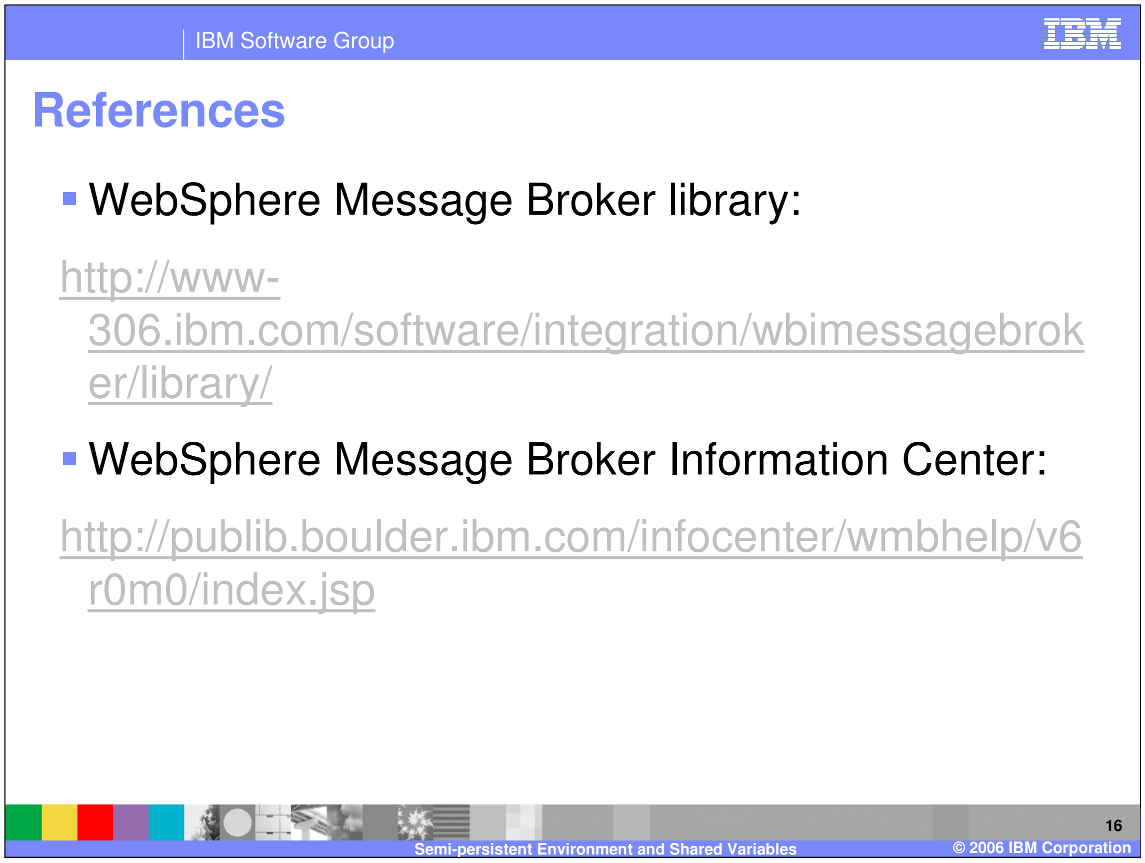
The last portion of the presentation contains a summary and references.



Summary

- Semi-persistent environment is new for Version 6
- The scope and lifetime of a shared variable is determined by where it is defined
- All threads within a respective scope can access a shared variable
- The new ROW data type provides greater programming flexibility for message tree data
- The ATOMIC block allows you to serialize setting and updating variables

This presentation discussed the semi-persistent environment, which is new for Version 6. Shared variables use this environment for storing and controlling access to their data. The scope and lifetime of a shared variable is determined by where the variable is defined. All threads within a respective scope can access a shared variable. In Version 6, a new ROW data type was introduced which can be used within normal or shared variable definitions, providing you greater programming flexibility for message tree data. For protecting the integrity of shared variables in a multi-threaded environment, you can use the ATOMIC block within your ESQL to serializing the setting or updating of the shared variables.



References

- WebSphere Message Broker library:

http://www-306.ibm.com/software/integration/wbimessagebroker/library/

- WebSphere Message Broker Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp

Template Revision: 04/25/2006 11:09 AM

Trademarks, copyrights, and disclaimers