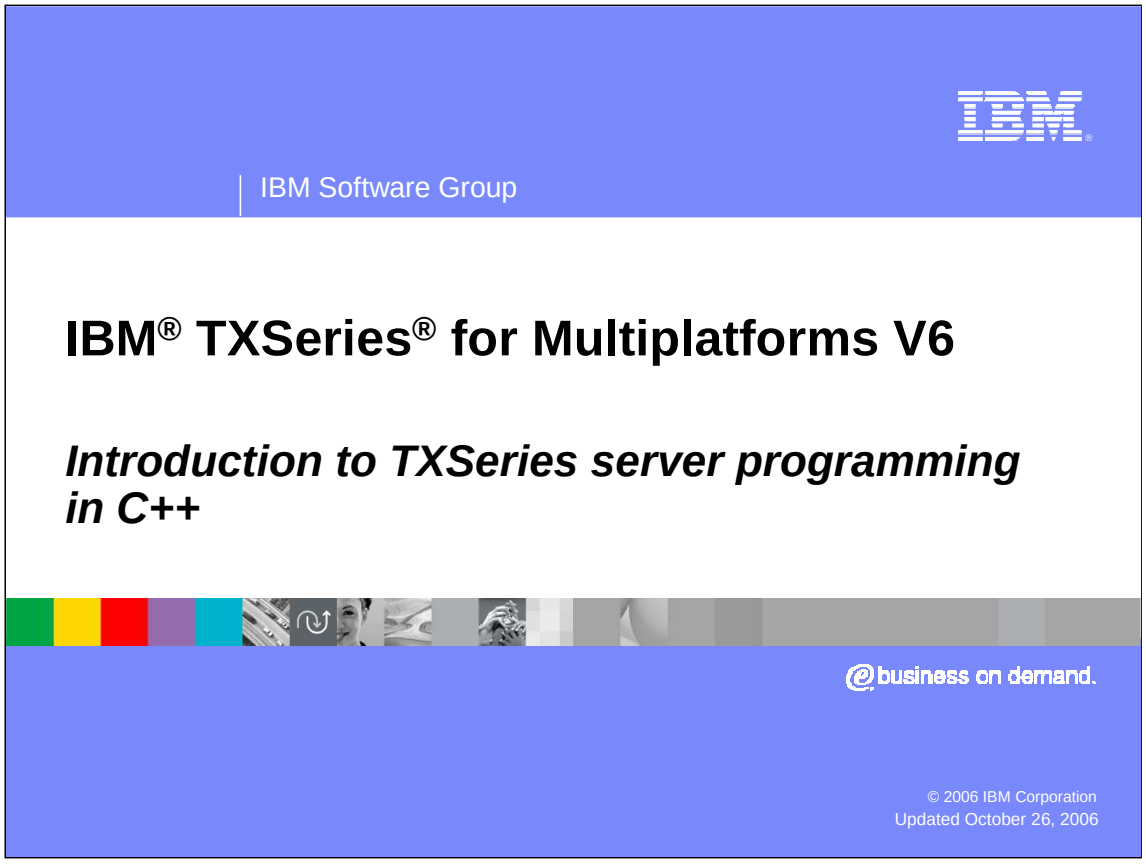


This presentation will introduce you to the C++ server programming language, one of the object oriented programming languages supported by TXSeries

Goals

- Provide an overview of object-oriented programming under TXSeries
- Components involved in writing C++ server applications
- Understanding CICS® foundation classes (CFC)
- CFC hierarchy

The goal of this presentation is to provide an overview of object-oriented programming as it relates to TXSeries.

Agenda

- Introduction to object oriented programming under TXSeries
  - Introduction to server programming in C++
    - Components involved
- CICS foundation classes
  - What is CFC?
  - CFC services
  - CFC class hierarchy

The agenda is to first provide an introduction to C++, followed by a definition of CFC and a description of the CFC services and the CFC class hierarchy.

Section

Introduction to object oriented programming under TXSeries

This section covers the support provided by TXSeries for writing server applications using C++ and the TXSeries components involved in the C++ language environment.

Object oriented programming under TXSeries

- TXSeries provides the flexibility of writing server programs using object-oriented programming style. With this support, you can:
  - Perceive TXSeries services as objects
  - Reusable classes
  - Create applications using object-oriented architecture.
  - Leverage use of existing application classes and tools provided by the language environment.

TXSeries provides the flexibility of writing server programs in both modular and object-oriented programming languages. With object-oriented programming, programmers can perceive TXSeries services as *objects*. TXSeries provides support for object-oriented programming in C++ and Java™. With this support, you can:

Perceive TXSeries services as objects.

Design and create reusable classes that can be used in and outside the TXSeries environment.

Design and create new modern TXSeries server applications using an object-oriented architecture.

Effectively use the strengths of various existing application classes and tools that are provided with the object-oriented language.

Server programming in C++ - Introduction

- TXSeries provides support for writing CICS server applications in the C++ programming language.
  - You can reuse existing application class libraries or templates and the rapid cross-platform development tools that are available to help write CICS server applications.
  - With C++, you can create reusable components quite easily, and use them in a CICS or non-CICS based systems (for example, batch applications).

TXSeries provides support for writing CICS server applications in the C++ programming language. You can reuse existing application class libraries or templates and the rapid cross-platform development tools that are available to help write CICS server applications. With C++, you can create reusable components quite easily, and use them in CICS or non-CICS based systems (for example, batch applications).

Components

- The following components are involved when writing CICS server programs using C++ programming language:
  - **C++ compiler**
  - **cicsprCpp**: C++ Language runtime
  - **CFC Library:** C++ Class library for CICS services

Three main components are involved when writing CICS server programs using the C++ programming language:

The first is the C++ compiler: TXSeries provides support for the Microsoft® C++ compiler on Windows®, and for IBM VisualAge C++ compiler on AIX®.

The second is the cicsprCpp, which is a C++ programming language runtime supplied by TXSeries. The runtime is loaded on to the application server process before

running any CICS server program that is compiled with C++ compiler.

The third is the CFC library, which provides various C++ class libraries with which applications can access CICS services.

Section

CICS foundation classes

This section covers the concept of CICS foundation classes (CFC) and the services offered by them.

CICS foundation classes (CFC)

- API for writing CICS server programs in C++
- Class libraries aid C++ and CICS development

The CICS foundation classes are an API for writing CICS server programs in the C++ programming language. They are well-designed class libraries that make C++ and CICS development much easier.

CICS foundation class library

- CICS foundation classes derived from the IccBase base class
  - enable common interfaces to be defined for categories of classes
  - create your own derived classes
  - Extending CICS foundation classes is simpler with an object-oriented approach
- Class categories:
  - Base and derived classes
  - Resource identification classes
  - Resource classes
  - Record index classes

All CICS foundation classes are derived, directly or indirectly, from the IccBase class. These classes enable common interfaces to be defined for categories of classes. You can also use them to create your own derived classes. Extending the functionality provided by the CICS foundation classes is much simpler through an object-oriented approach. In general, the following class categories are defined:

Base and derived classes are the fundamental classes that are derived from the IccBase class, such as IccException for exception

handling and IccBuf for manipulating buffers.

Resource identification classes model the behavior of major CICS resources such as IccTerminal for Terminal, IccProgram for

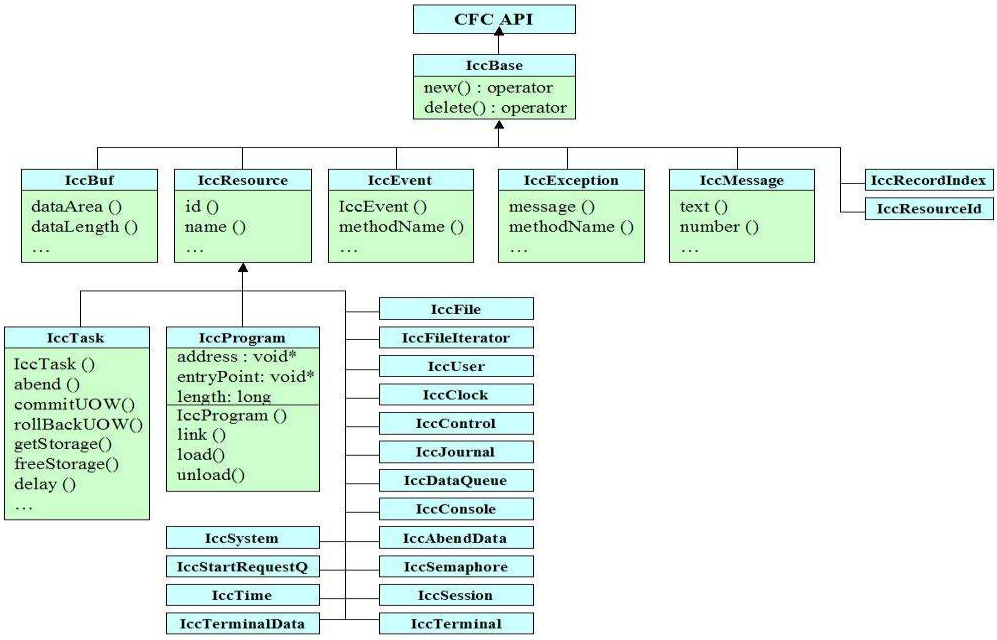
Programs, and IccTempStore for temporary storage queues.

Resource classes help to define CICS resource identifiers such as FD, PD, and TD stanza entries. These classes improve type

checking. For example, methods that expect an IccField object as a parameter do not accept an IccProgramId object instead.

Record index classes are used to identify records in CICS files such as IccKey for a KSDS file, and IccRRN for a RRDS file. They specify the record access method and identify which record in a file is to be retrieved, modified, or deleted.

JCICS class hierarchy



The Figure in this slide illustrates the JCICS class hierarchy.

Summary

- An overview of object-oriented programming under TXSeries
- Components involved in writing C++ server applications
- Understanding CICS foundation classes (CFC)
- CFC hierarchy

In summary, this presentation covered C++ for TXSeries server programming, including C++ server applications, the CICS foundation classes, and the CICS foundation class hierarchy.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.