The CICS Asynchronous API

Leigh Compton

IBM Washington Systems Center
lcompton@us.ibm.com

Please note

Abstract

Learn how the CICS Asynchronous API can help you reduce your application's response time. Available in the CICS TS 5.4 the Asynchronous API commands can help your applications become more robust, and allow you to make better use of your time.

Allow CICS to simplify your asynchronous processing mechanisms and break away from polling code to get better results with a first-class API.

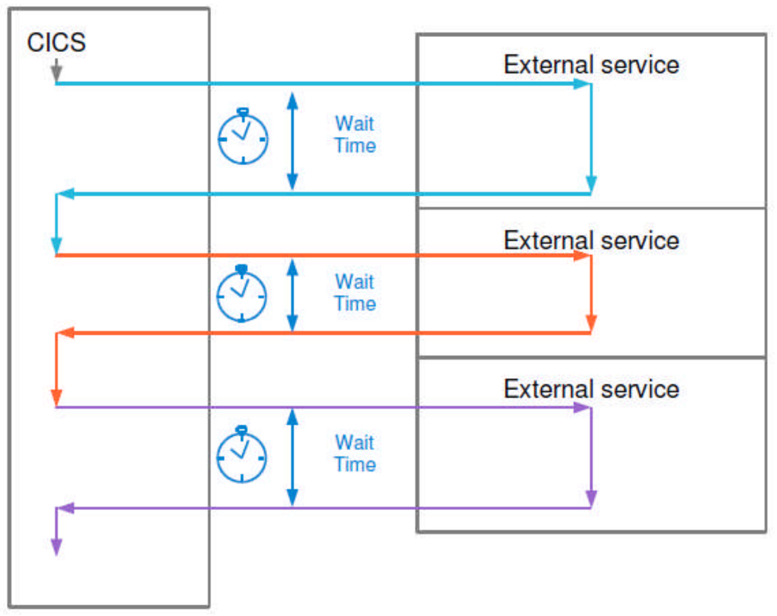
Reduce the risk of unpredictable external services by calling multiple services and consuming the responses as soon as they arrive..

Act 1
What customers are trying to do

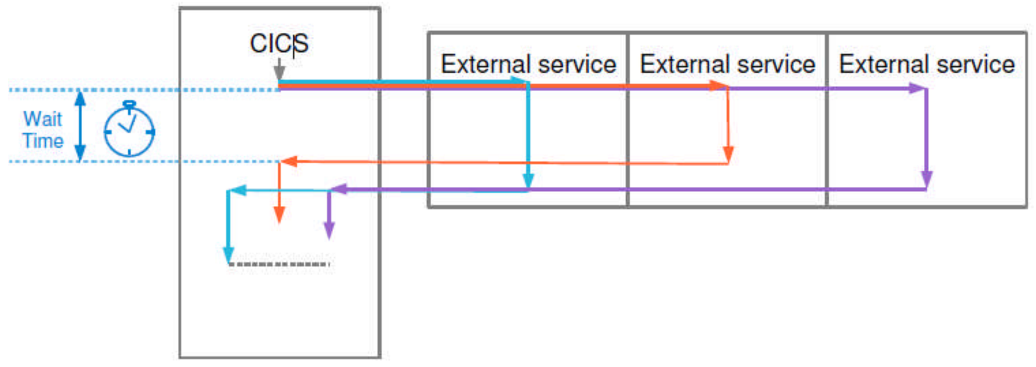
Make better use of your time

Can you do something more useful than waiting around for a service call to return?

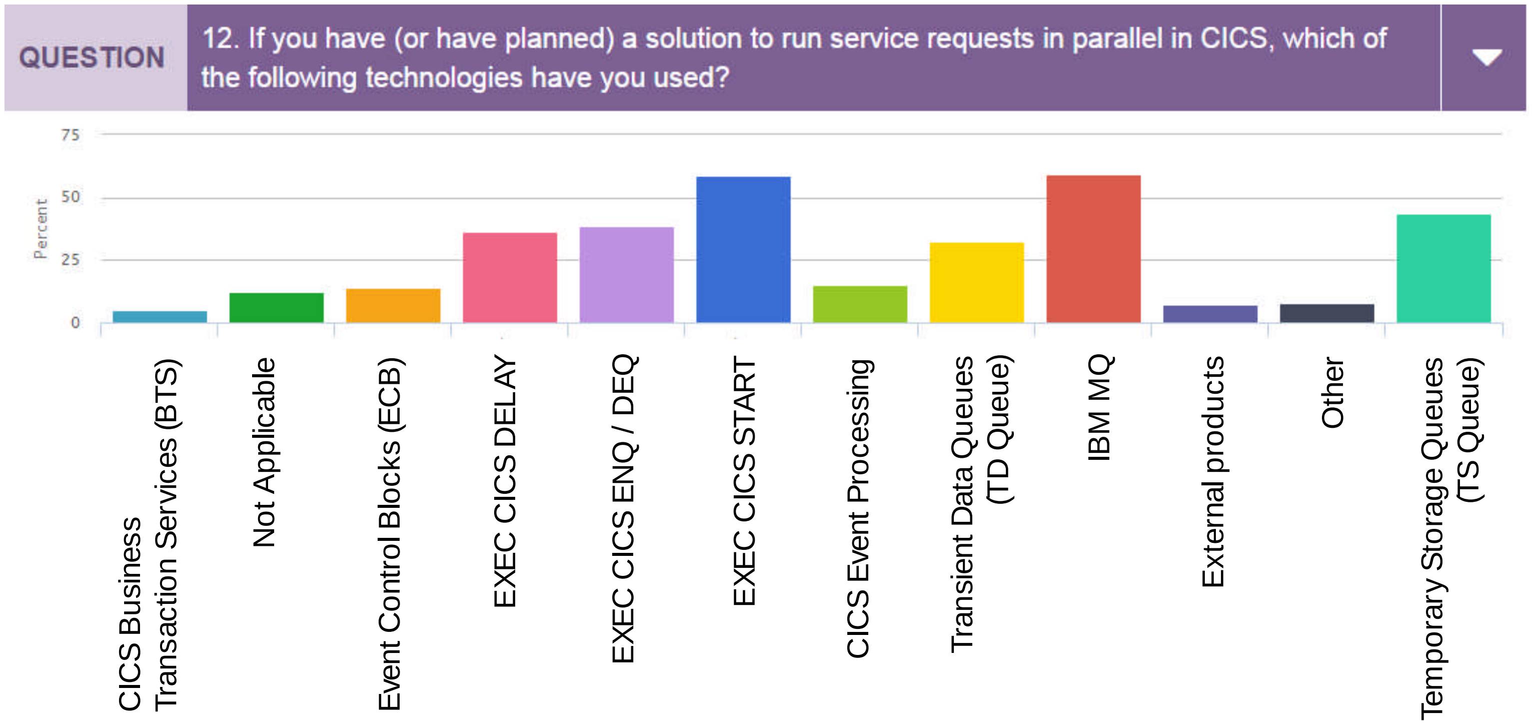
From this:



To this:

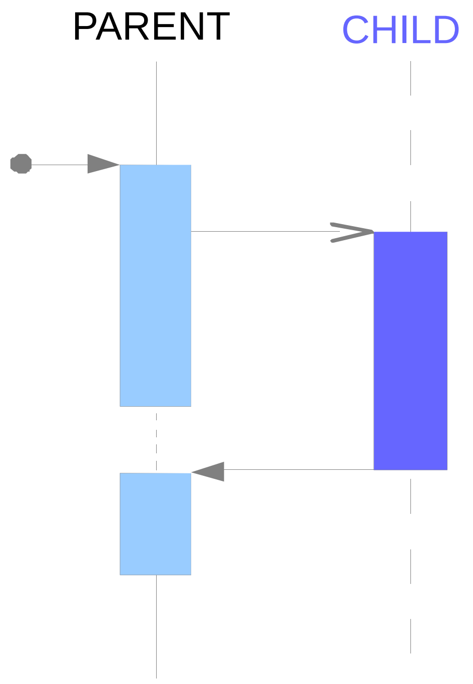


How are customers doing this today with CICS?


QUESTION 12. If you have (or have planned) a solution to run service requests in parallel in CICS, which of the following technologies have you used?

Act 2
A Better Solution

New supported set of CICS Asynchronous API’s



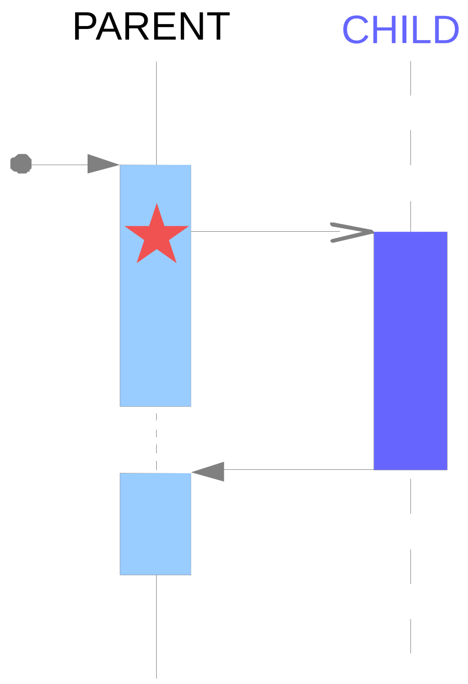
Key challenges

1. Run transaction asynchronously
2. Fetch child completion
3. Pass data safely

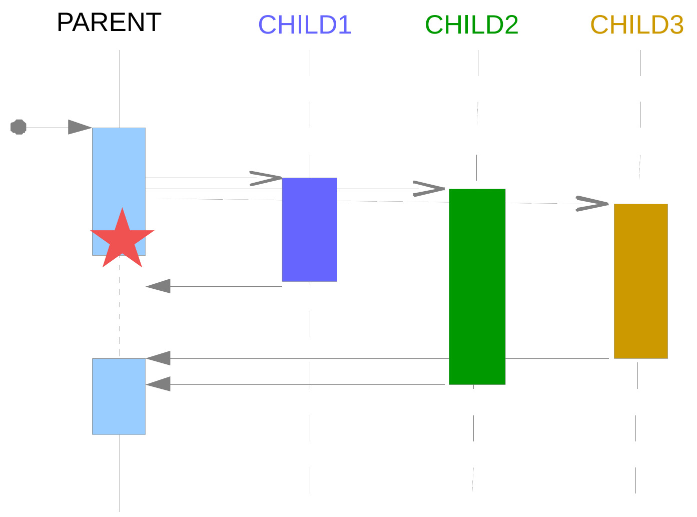
Run transactions asynchronously

```
RUN TRANSID (transaction)
  CHILD     (identifier)
```

“I need a credit check but I can get on with some other work while I wait for the results”



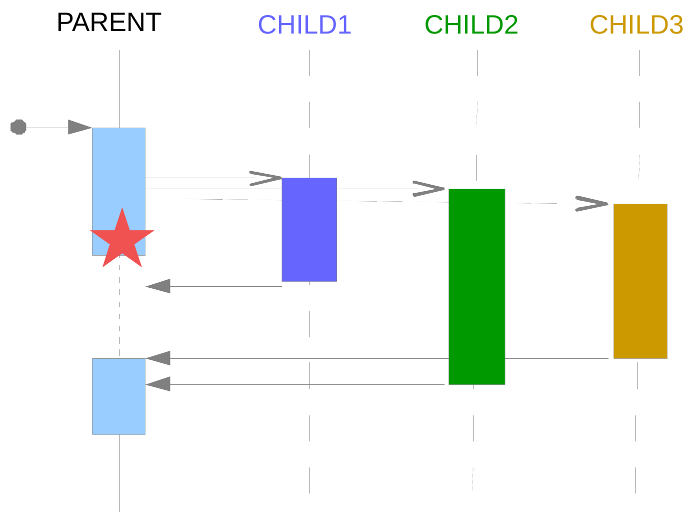
Fetch child completion

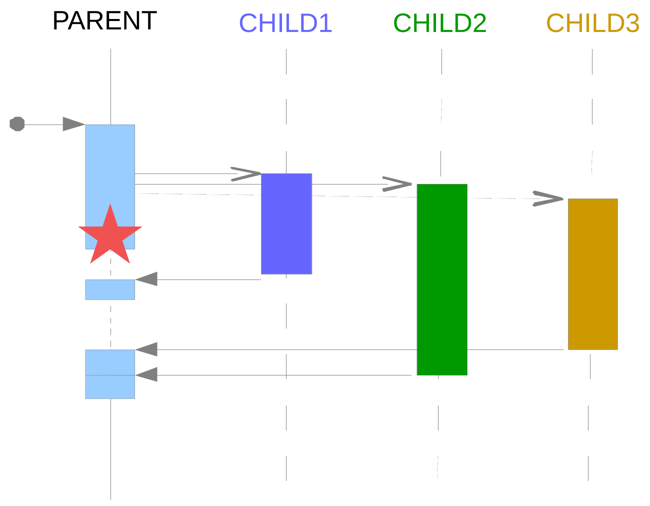


```
FETCH CHILD(in-identifier)
```

“I cannot continue processing the parent coordinator, until I have received confirmation from CHILD3”

Fetch child completion – Two new commands





`FETCH CHILD(in-identifier)`

“I cannot continue processing the parent coordinator, until I have received confirmation from CHILD3”

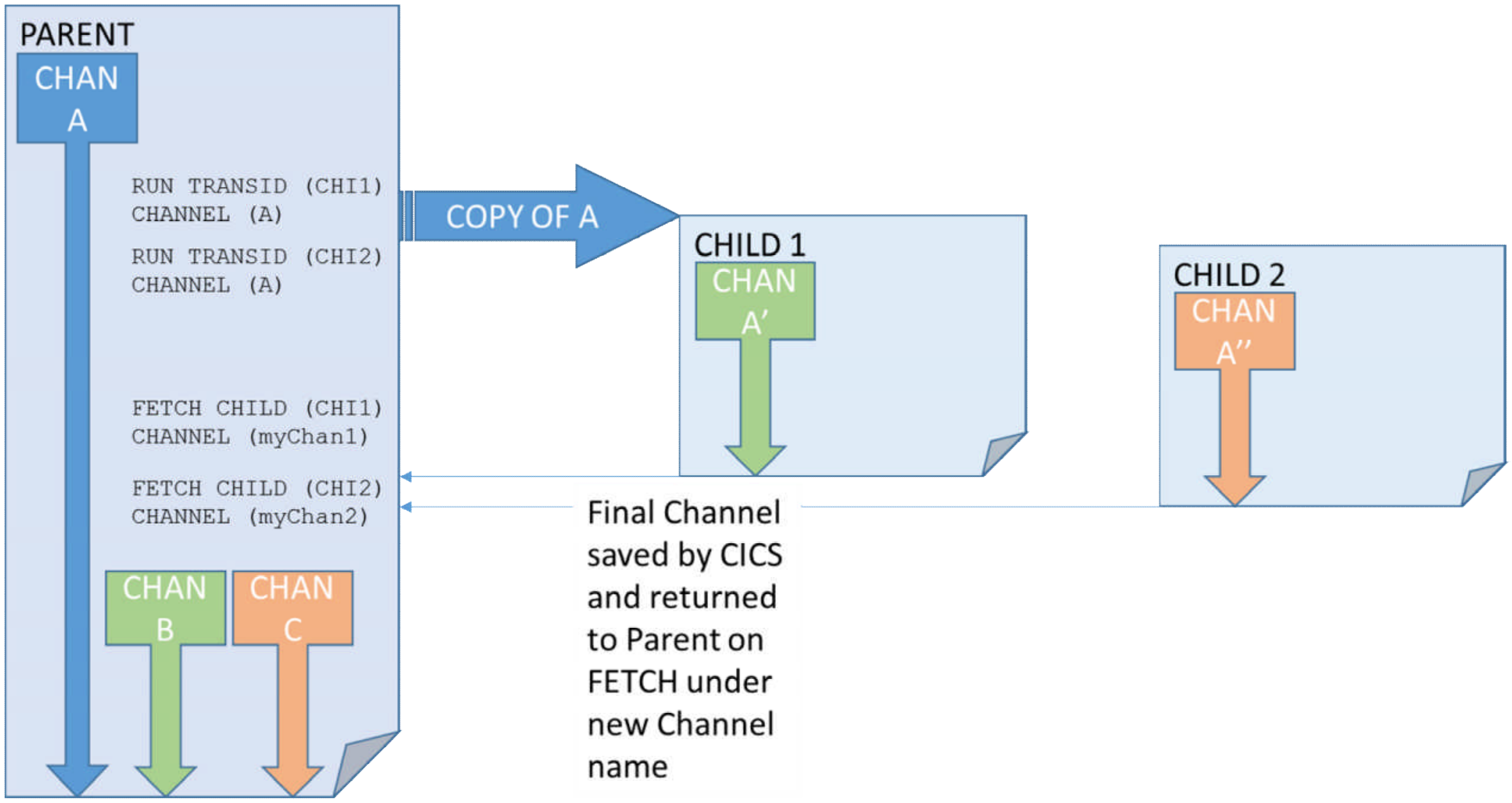
`FETCH ANY(out-identifier)`

“I only need the first one to reply”

Passing Data

CICS Channels and Containers

(not COMMAREA!)


PARENT
CHAN A
RUN TRANSID (CHI1) CHANNEL (A)
RUN TRANSID (CHI2) CHANNEL (A)
FETCH CHILD (CHI1) CHANNEL (myChan1)
FETCH CHILD (CHI2) CHANNEL (myChan2)
CHAN B
CHAN C
COPY OF A
CHILD 1
CHAN A’
CHILD 2
CHAN A’’
Final Channel saved by CICS and returned to Parent on FETCH under new Channel name

Technical Stuff:
The programming interface

RUN TRANSID

```
RUN TRANSID

>>-RUN--TRANSID(name)--+----------------+--CHILD(data-area)----><
                       '-CHANNEL(name)-'
```

RUN TRANSID

RUN TRANSID initiates a local child transaction that runs asynchronously with the parent transaction.

RUN TRANSID starts a task in the local CICS region, optionally passing a channel.

The child task inherits the security context of the parent task.

CHANNEL(name)

- specifies the name of a channel that is to be made available to the child task
- child is given a copy of the channel's containers

CHILD(data-area)

- specifies a 16-character binary data area into which CICS will place the child token.

TRANSID(name)

- specifies the local transaction to be started

FETCH CHILD

```
FETCH CHILD

>>-FETCH--CHILD(data-value)--+-------------------+------------->
                             '-CHANNEL(data-area)-'

>--COMPSTATUS(cvda)--+------------------+---------------------->
                     '-ABCODE(data-area)-'

>--+---------------------+------------------------------------><
   +-NOSUSPEND-----------+
   '-TIMEOUT(data-value)-'
```

FETCH CHILD

FETCH CHILD command is used by a parent task to inquire on the status of a specific child task.

FETCH CHILD returns the status of the specified child task.

CHILD(data-value)

- specifies the child token to fetch the response from

COMPSTATUS(cvda)

- Returns a cvda indication completion status of the child task

CHANNEL(data-area)

- returns the 16-character name of the child task's reply channel.
- channel name is generated by CICS

TIMEOUT(data-value)

- specifies a fullword binary data value which is the maximum time in milliseconds that the command will wait for the child to complete.

NOSUSPEND

- Indicates that the command will return immediately without waiting for the child task to complete.

FETCH ANY

```
FETCH ANY

>>-FETCH--ANY(data-area)--+---------------------+---------------->
                          '-CHANNEL(data-area)-'

>--COMPSTATUS(cvda)--+-------------------+----------------------->
                     '-ABCODE(data-area)-'

>--+---------------------+--------------------------------------><
   +-NOSUSPEND-----------+
   '-TIMEOUT(data-value)-'
```

FETCH ANY

FETCH CHILD command is used by a parent task to inquire on the status of any child task.

FETCH CHILD returns the status of any completed child task which has not yet been fetched.

ANY(data-value)

- specifies a 16-character binary data area into which CICS will place the fetched child token.

COMPSTATUS(cvda)

- Returns a cvda indication completion status of the child task

CHANNEL(data-area)

- returns the 16-character name of the child task's reply channel.
- channel name is generated by CICS

TIMEOUT(data-value)

- specifies a fullword binary data value which is the maximum time in milliseconds that the command will wait for any remaining child to complete.

NOSUSPEND

- Indicates that the command will return immediately without waiting for the child task to complete.

FREE CHILD

```
FREE CHILD

>>-FREE--CHILD(data-area)----------------------------><
```

FREE CHILD

FREE CHILD command frees a specified child token

Using the FREE CHILD command will free the resources associated with the child task when it completes, rather than waiting for them to be fetched.

CICS implicitly frees all child tokens when the parent task terminates.

CHILD(data-value)

- specifies the child token to be freed.

JCICS classes

Not just a simple port!

Implements java.util.concurrent.Future interface

Three classes encompass all of EXEC CICS commands:

- AsyncService
- AsyncServiceImpl
- ChildResponse

AsyncService

    runTransactionID()

    freeChild()

ChildResponse

    getAny()

Future<ChildResponse>

    get()

    isDone()

JCICS example

```
final String childTransaction = “ABCD”;
AsyncService myAsync = new AsyncServiceImpl();

Future<ChildResponse> myChild = myAsync.runTransactionID(childTransaction);

// Logic here to be processed while the child runs asynchronously

ChildResponse myResponse = myChild.get();
if (myResponse.getCompletionStatus().equals(CompletionStatus.NORMAL)) {
    System.out.println("Child task completed normally");
}
```

Act 3

The complete solution

What if my services are unreliable?

“How can I tell if the child completed successfully?”

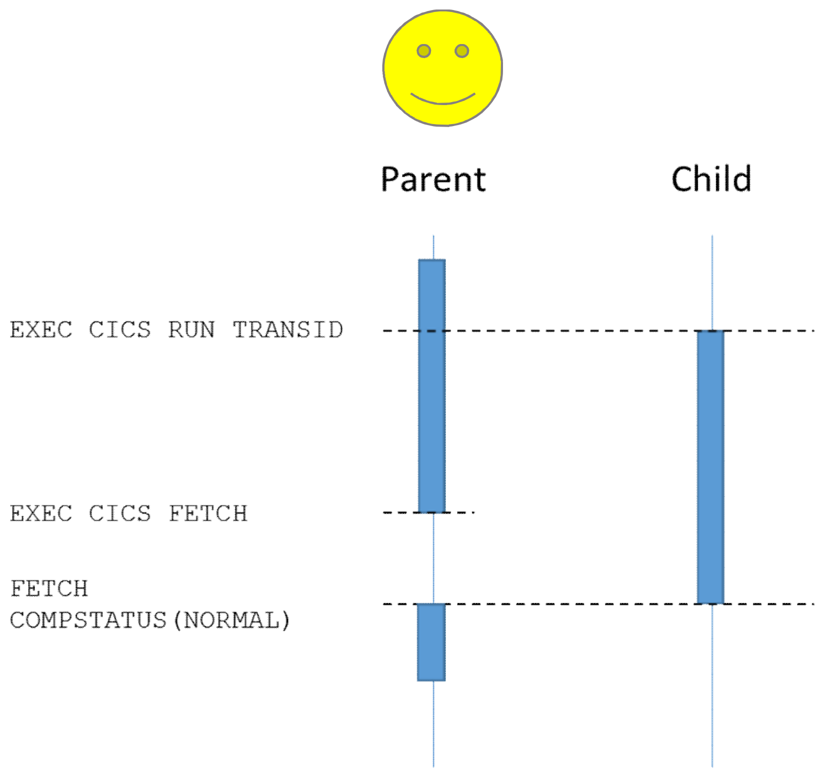
The completion status is a first class parameter on the FETCH API

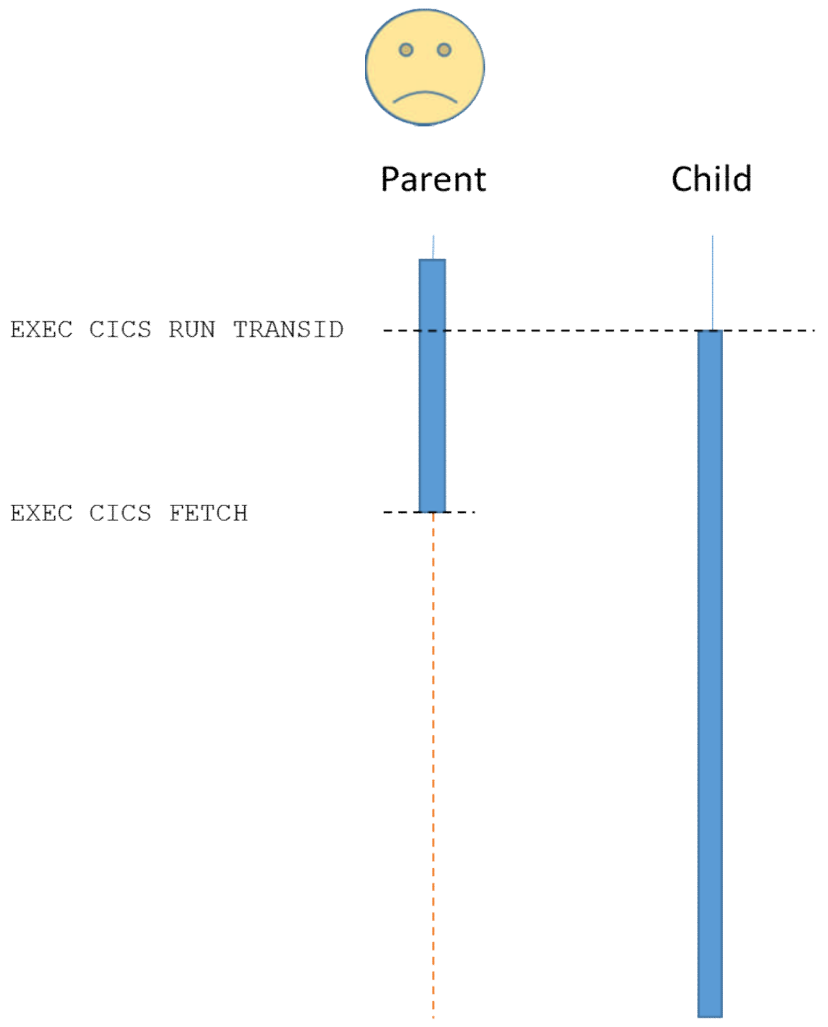
How can I tell if my child completed successfully?

```
FETCH CHILD
 >>-FETCH--CHILD(data-area)--+------------------+---------------->
                             '-CHANNEL(data-area)-'
 >--COMPSTATUS(cvda)--+-------------------+---------------------->
                      '-ABCODE(data-area)-'
 >--+----------------------+------------------------------------><
    +-NOSUSPEND-----------+
    '-TIMEOUT(data-value)-'
```

- COMPSTATUS returns a CVDA indicating the child task's completion status:
  - ABEND
  - NORMAL
  - SECERROR
- RESP=NOTFINISHED

Don’t want to wait forever!



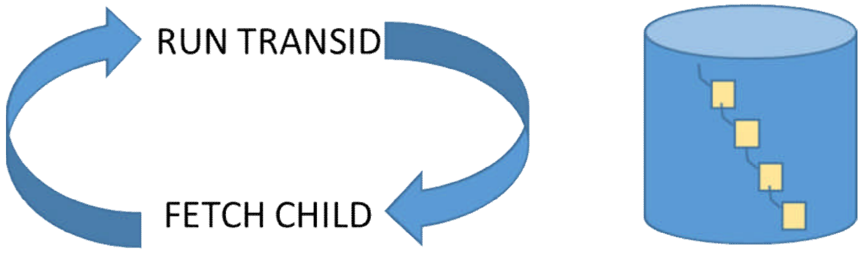


Don’t want to wait forever!

```
FETCH ANY
 >>-FETCH--ANY(data-area)--+--------------------+---------------->
                           '-CHANNEL(data-area)-'
 >--COMPSTATUS(cvda)--+------------------+---------------------->
                      '-ABCODE(data-area)-'
 >--+---------------------+------------------------------------><
    +-NOSUSPEND-----------+
    '-TIMEOUT(data-value)-'
```

* TIMEOUT(0) means that timeout is not being set

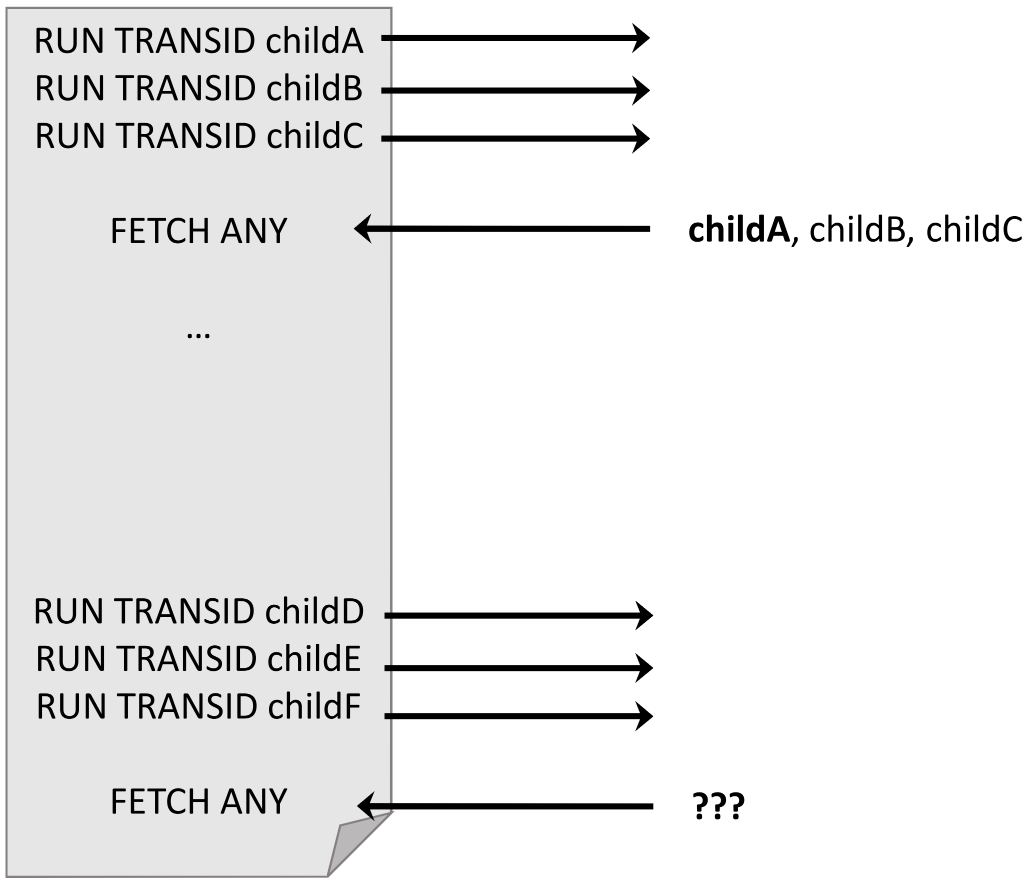
FREE CHILD: Useful for Long-Running Parents



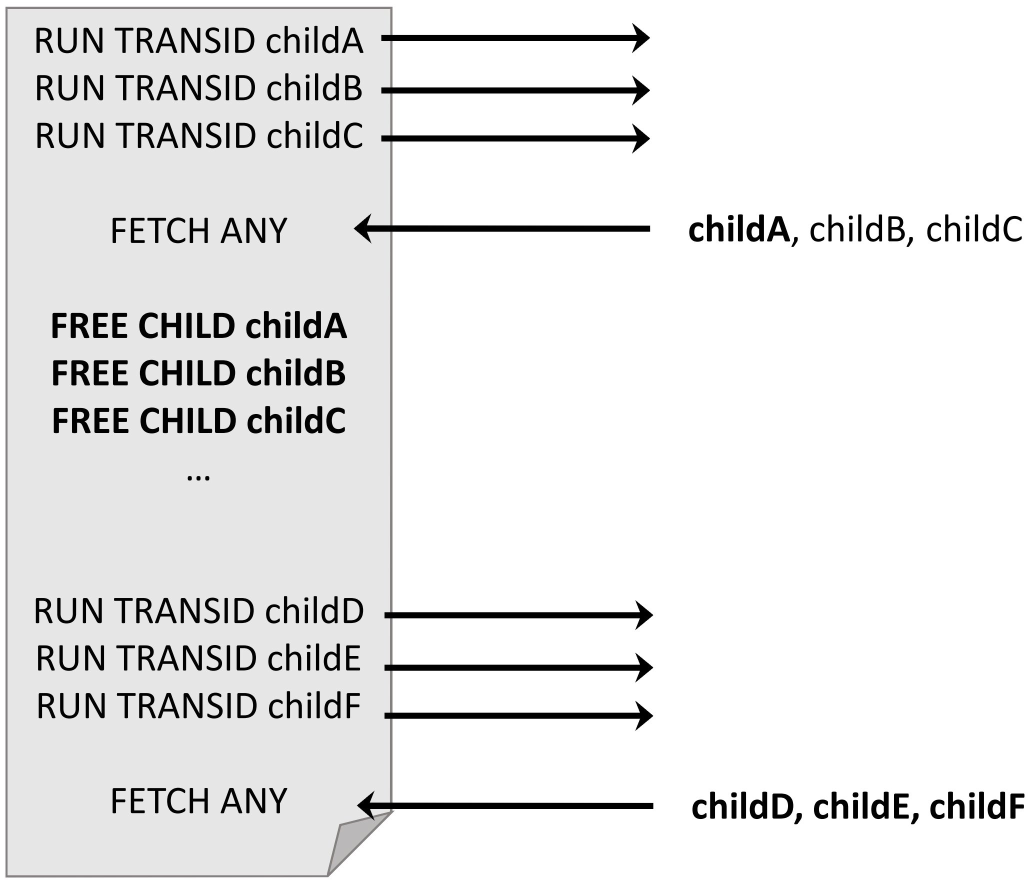
Ability to disassociate parent from child

Application can DELETE CHANNEL for already FETCHed results

FREE CHILD: Batch Service Calls



FREE CHILD: Batch Service Calls



CICS Automated Control

Emergency Brake

- Control dispatching of asynchronous tasks
- Protect CICS rather than to optimise throughput
- Prevent flooding the CICS system
- Solely in CICS's control

Transaction Classes

Control your systems!

DO have transaction classes

DO NOT put parents and children in the same transaction class

CICS Policies

Control your systems!

Task rule condition:

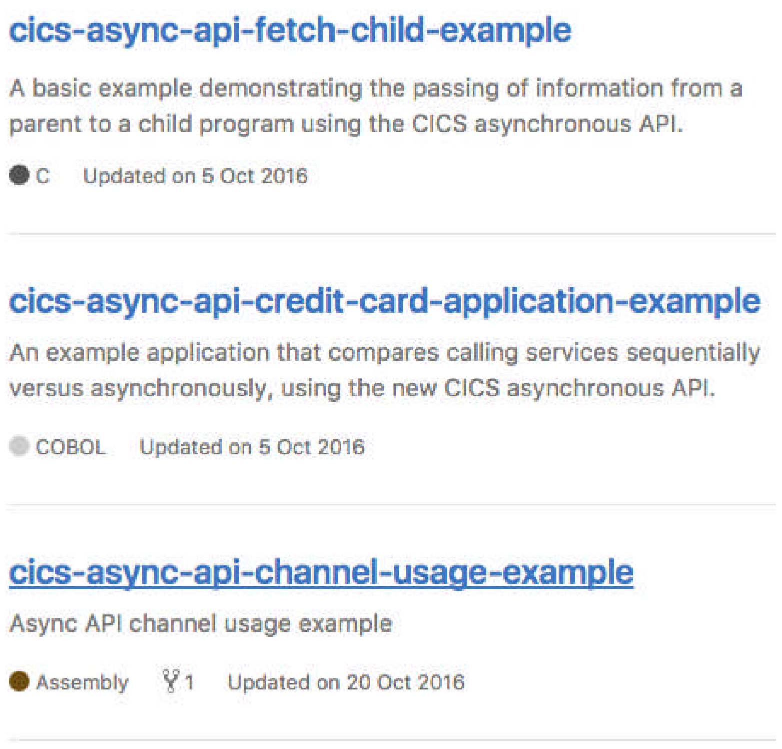
- RUN TRANSID
  - Threshhold

Policy actions:

- issue message
- emit event
- abend task

Articles, Knowledge Center, Example Code

github.com/**cicsdev**



cics-async-api-fetch-child-example

A basic example demonstrating the passing of information from a parent to a child program using the CICS asynchronous API.

● C Updated on 5 Oct 2016

cics-async-api-credit-card-application-example

An example application that compares calling services sequentially versus asynchronously, using the new CICS asynchronous API.

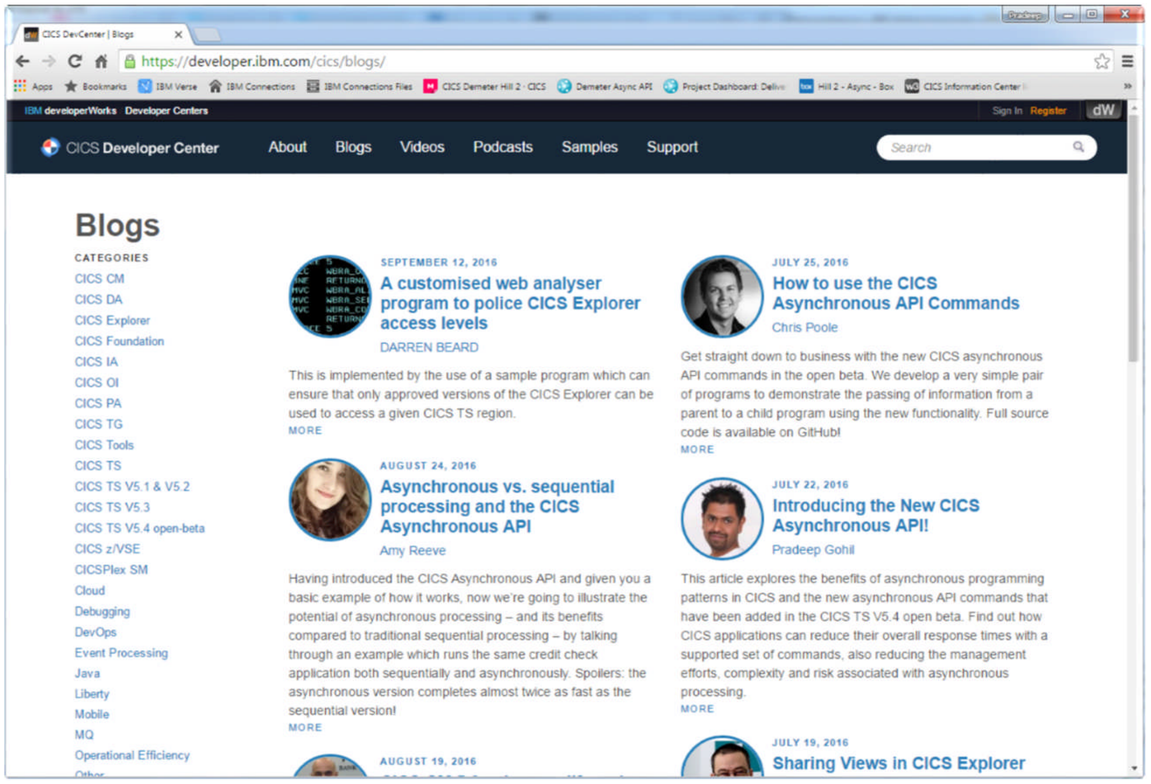
● COBOL Updated on 5 Oct 2016

cics-async-api-channel-usage-example

Async API channel usage example

● Assembly 1 Updated on 20 Oct 2016

developer.ibm.com/**cics**



Blogs

CATEGORIES

CICS CM
CICS DA
CICS Explorer
CICS Foundation
CICS IA
CICS OI
CICS PA
CICS TG
CICS Tools
CICS TS
CICS TS V5.1 & V5.2
CICS TS V5.3
CICS TS V5.4 open-beta
CICS z/VSE
CICSPlex SM
Cloud
Debugging
DevOps
Event Processing
Java
Liberty
Mobile
MQ
Operational Efficiency

SEPTEMBER 12, 2016
A customised web analyser program to police CICS Explorer access levels
DARREN BEARD
This is implemented by the use of a sample program which can ensure that only approved versions of the CICS Explorer can be used to access a given CICS TS region.
MORE

JULY 25, 2016
How to use the CICS Asynchronous API Commands
Chris Poole
Get straight down to business with the new CICS asynchronous API commands in the open beta. We develop a very simple pair of programs to demonstrate the passing of information from a parent to a child program using the new functionality. Full source code is available on GitHub!
MORE

AUGUST 24, 2016
Asynchronous vs. sequential processing and the CICS Asynchronous API
Amy Reeve
Having introduced the CICS Asynchronous API and given you a basic example of how it works, now we're going to illustrate the potential of asynchronous processing – and its benefits compared to traditional sequential processing – by talking through an example which runs the same credit check application both sequentially and asynchronously. Spoilers: the asynchronous version completes almost twice as fast as the sequential version!
MORE

JULY 22, 2016
Introducing the New CICS Asynchronous API!
Pradeep Gohil
This article explores the benefits of asynchronous programming patterns in CICS and the new asynchronous API commands that have been added in the CICS TS V5.4 open beta. Find out how CICS applications can reduce their overall response times with a supported set of commands, also reducing the management efforts, complexity and risk associated with asynchronous processing.
MORE

AUGUST 19, 2016

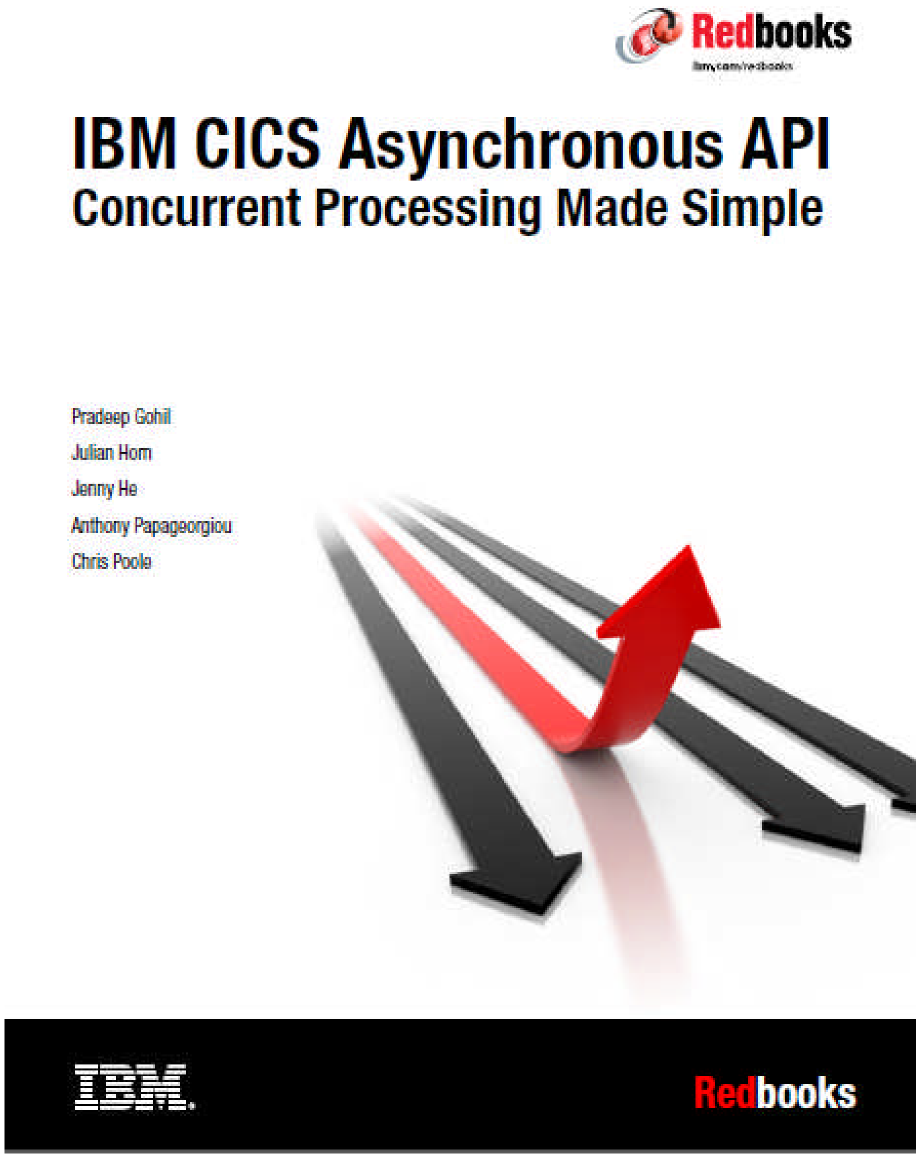
JULY 19, 2016
Sharing Views in CICS Explorer

IBM CICS Asynchronous API: Concurrent Processing Made Simple

Redbook SG24-8411

http://www.redbooks.ibm.com/abstracts/sg248411.html?Open

This IBM® Redbooks® publication covers the background and implementation of the IBM CICS® asynchronous API, which is a simple, accessible API that is designed to enable CICS application developers to create efficient asynchronous programs in all CICS-supported languages. Using the API, application developers can eliminate the overhead that is involved in coding and managing homegrown asynchronous solutions

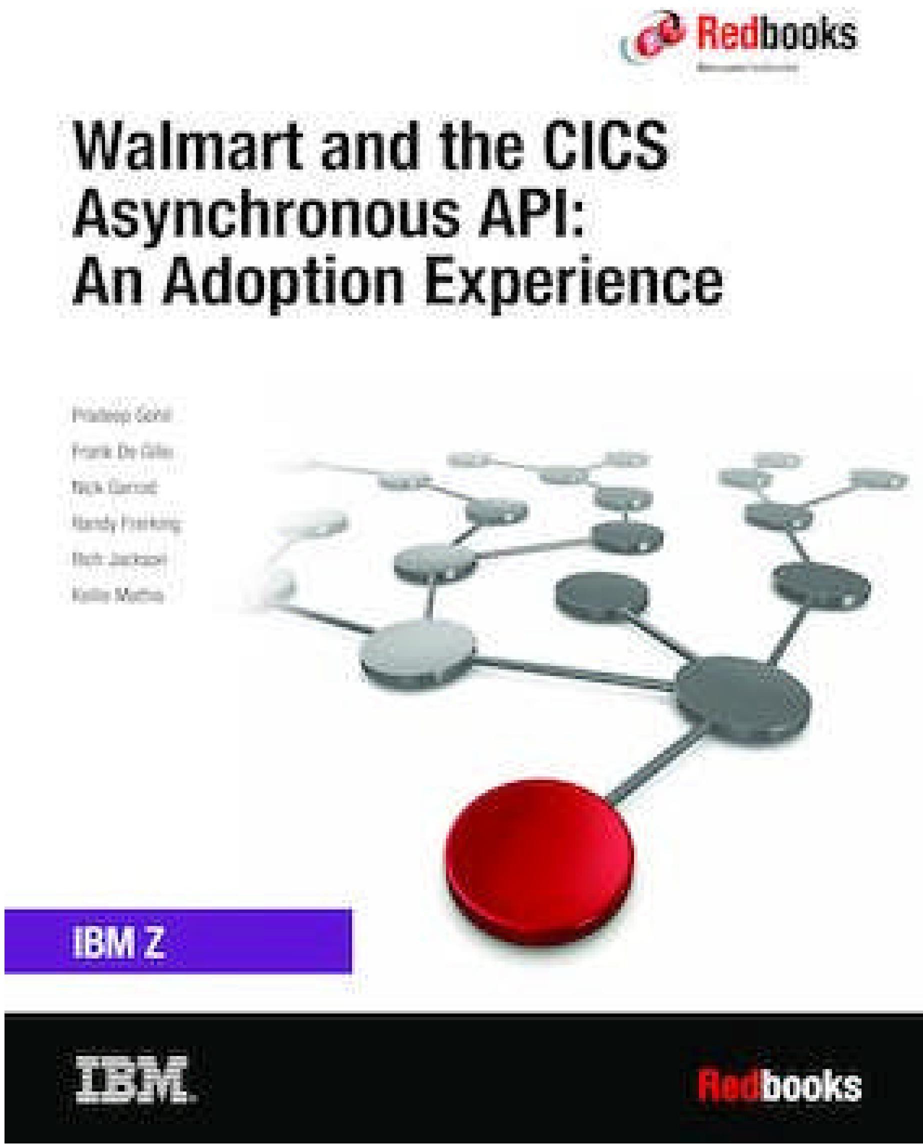


Walmart and the CICS Asynchronous API

Redbook SG24-8444

http://www.redbooks.ibm.com/abstracts/sg248444.html?Open

This IBM® Redbooks® publication discusses practical uses of the IBM CICS asynchronous API capability. It describes the methodology, design and thought process used by a large client, Walmart, and the considerations of the choices made. The Redbooks publication provides real life examples and application patterns that benefit from the performance and scalability offered by the new API.



Redbooks

Walmart and the CICS Asynchronous API: An Adoption Experience

Pradeep Gohil
Frank De Gilio
Nick Garrod
Randy Frerking
Rich Jackson
Kellie Mathis

IBM Z

IBM

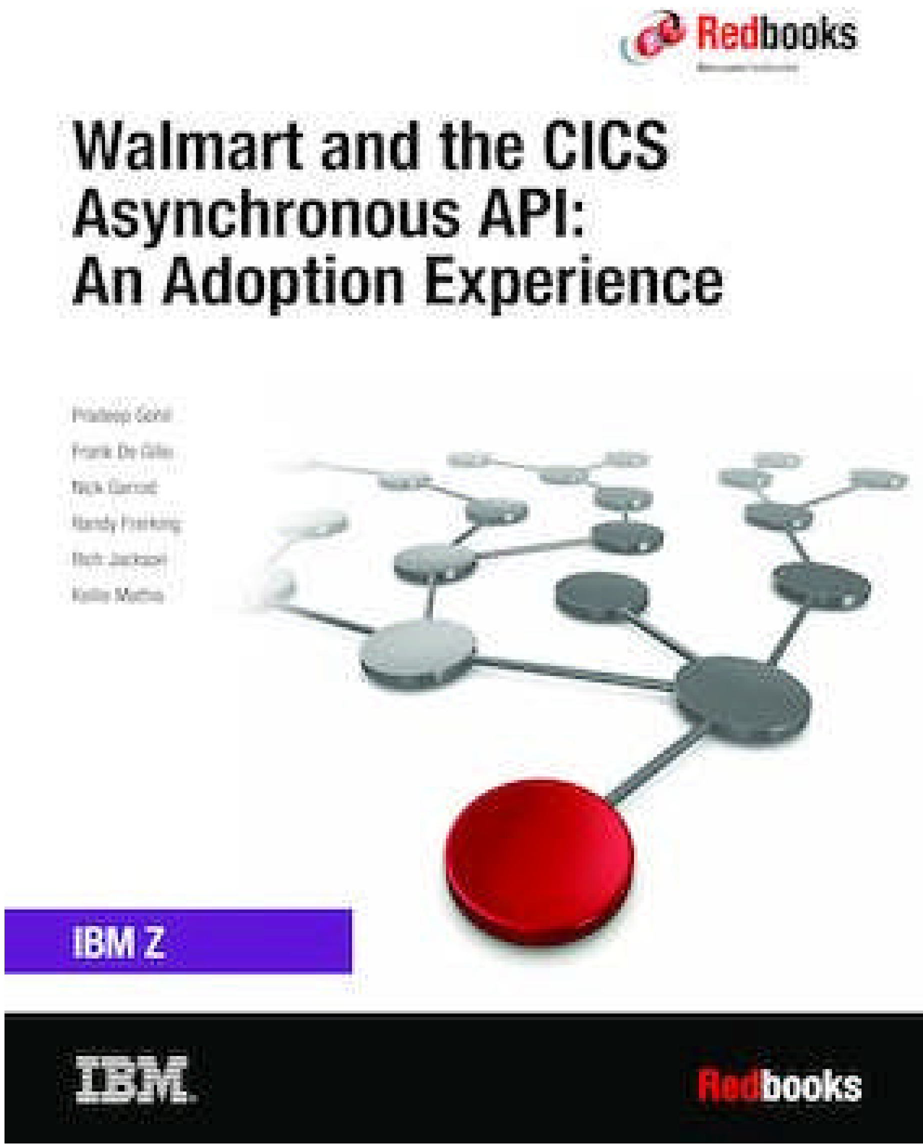
Redbooks

Walmart and the CICS Asynchronous API

Video course CRSE-0306

http://www.redbooks.ibm.com/abstracts/crse0306.html?Open

In this video series we follow the experience of the Walmart delivery team as they embrace asynchronous processing patterns in enterprise-grade applications. The series describes the methodology, design and thought process used by Walmart and the considerations of the choices made. By using IBM CICS asynchronous API, Walmart enhanced a complex search capability to achieve large scale transactions in minimal time.



Redbooks

Walmart and the CICS Asynchronous API: An Adoption Experience

Pradeep Gohil
Frank De Gilio
Nick Garrod
Randy Frerking
Rich Jackson
Kellie Mathis

IBM Z

IBM

Redbooks

Thank you

Notices and disclaimers

Notices and disclaimers continued